

# Arm China Cortex®-M52 Processor

Revision: r0p2

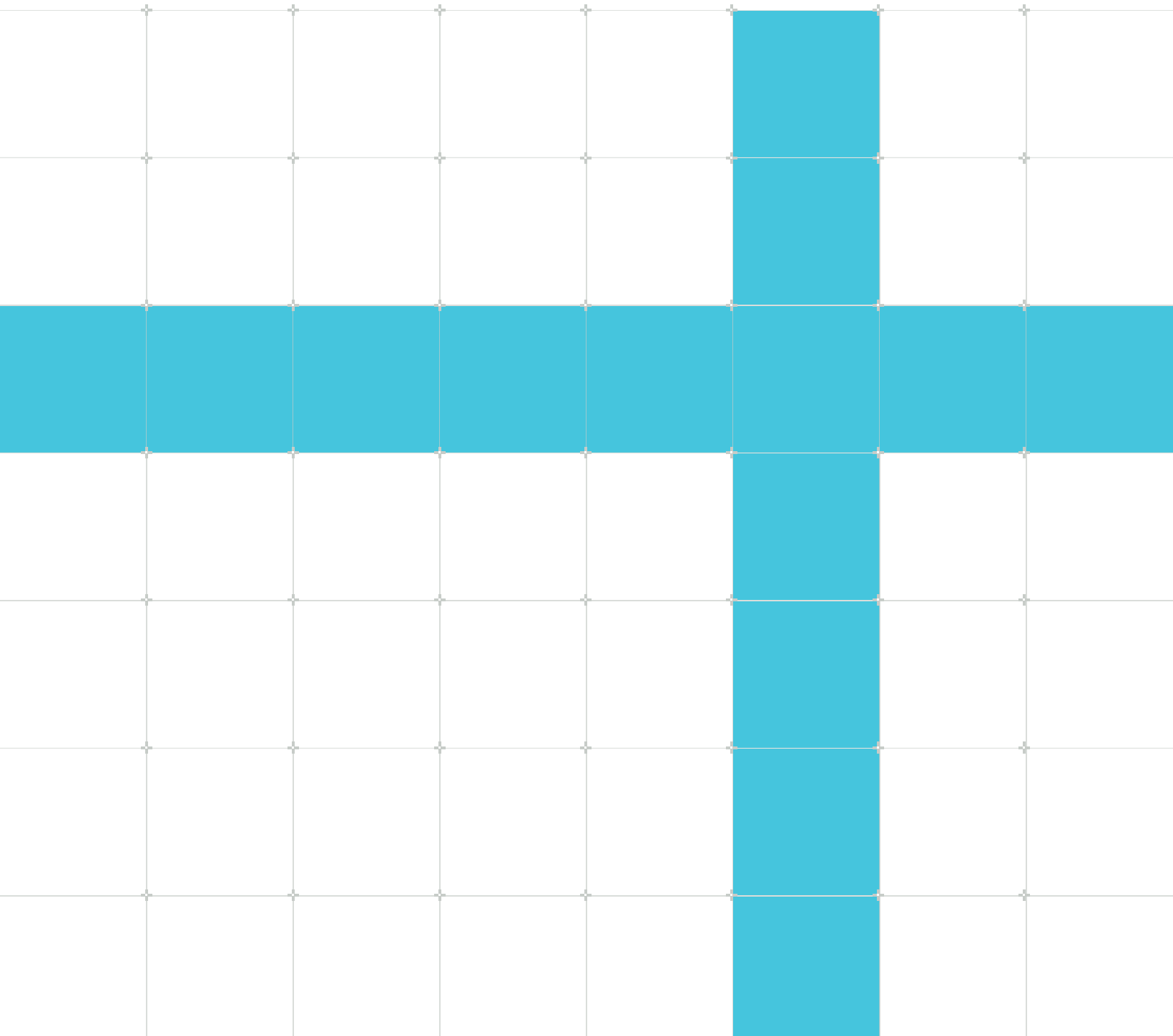
## Technical Reference Manual

**Non-Confidential**

Copyright © 2022–2023 Arm Technology (China) Co., Ltd. (or its affiliates) and Copyright © 2019–2021 Arm Limited (or its affiliates). All rights reserved.

**Issue 06**

102776\_0002\_06\_en



# Arm China Cortex®-M52 Processor

## Technical Reference Manual

Copyright © 2022–2023 Arm Technology (China) Co., Ltd. (or its affiliates) and Copyright © 2019–2021 Arm Limited (or its affiliates). All rights reserved.

## Release Information

### Document history

Issue	Date	Confidentiality	Change
0000-01	15 January 2022	Confidential	First beta release for r0p0
0000-02	24 April 2022	Confidential	First limited access release for r0p0
0001-03	27 July 2022	Non-Confidential	First early access release for r0p1
0001-04	30 September 2022	Non-Confidential	Second early access release for r0p1
0002-05	30 December 2022	Non-Confidential	First release for r0p2
0002-06	30 September 2023	Non-Confidential	Second release for r0p2

## Proprietary Notice

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

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

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

This document may include technical inaccuracies or typographical errors.

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

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

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

Arm China is a trading name of Arm Technology (China) Co., Ltd. The words marked with ® or ™ are registered trademarks or trademarks of Arm Limited (or its affiliates) in the People's Republic of China and/or elsewhere. All rights reserved. Visit <https://www.arm.com/company/policies/trademarks> and <https://www.armchina.com/usestandard> for full guidance on using Arm's trademarks. Other brands and names mentioned in this document may be the trademarks of their respective owners.

Copyright © 2022-2023 Arm Technology (China) Co., Ltd. (or its affiliates).

Copyright © 2019-2021 Arm Limited (or its affiliates). All rights reserved.

Arm Limited. Company 02557590 registered in England.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

Arm Technology (China) Co., Ltd. registered in China.

Room 201, Building A, No. 1 First Qianwan Road, Qianhai Shengang Cooperation Zone, Shenzhen, the People's Republic of China.

(LES-PRE-20349 - Arm China)

## Confidentiality Status

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

Unrestricted Access is an Arm China internal classification.

## Product Status

The information in this document is Final, that is for a developed product.

# Contents

<b>1. Introduction.....</b>	<b>17</b>
1.1 Product revision status.....	17
1.2 Intended audience.....	17
1.3 Conventions.....	17
1.4 Useful resources.....	19
<b>2. The Cortex®-M52 processor.....</b>	<b>21</b>
2.1 Cortex®-M52 processor overview.....	21
2.2 Cortex®-M52 features.....	22
2.3 Supported standards and specifications.....	24
2.4 Design tasks.....	26
2.5 Documentation.....	27
2.6 Product revisions.....	28
<b>3. Technical overview.....</b>	<b>29</b>
3.1 Cortex®-M52 processor components.....	29
3.1.1 Cortex®-M52 processor core.....	31
3.1.2 Extension Processing Unit.....	32
3.1.3 Memory components.....	33
3.1.4 Interrupt components.....	36
3.1.5 Debug and trace components.....	37
3.1.6 Testing components.....	38
3.2 Interfaces.....	38
3.3 Security.....	39
3.4 Functional safety and reliability.....	40
3.5 Power intent.....	41
3.6 Performance considerations.....	41
3.7 Cortex®-M52 implementation options.....	41
<b>4. Programmers model.....</b>	<b>44</b>
4.1 Security states, operation, and execution modes.....	44
4.2 Instruction set summary.....	45
4.3 Exclusive monitor.....	46

4.4 Cortex®-M52 processor core registers summary.....	46
4.5 Architectural registers.....	48
4.6 Exceptions.....	49
<b>5. System registers.....</b>	<b>51</b>
5.1 System control register summary.....	51
5.2 Identification register summary.....	55
5.2.1 Media and VFP Feature Register reset values, MVFR0, MVFR1, and MVFR2 reset values....	59
5.3 AFSR, Auxiliary Fault Status Register.....	59
5.4 CPUID, CPUID Base Register.....	61
5.5 ID_AFR0, Auxiliary Feature Register 0.....	62
5.6 Cache identification register summary.....	63
5.6.1 CLIDR, Cache Level ID Register.....	64
5.6.2 CSSELR, Cache Size Selection Register.....	65
5.6.3 CCSIDR, Current Cache Size ID Register.....	66
5.7 REVIDR, Revision ID Register.....	68
5.8 Implementation control register summary.....	69
5.9 ACTLR, Auxiliary Control Register.....	69
5.10 ICTR, Interrupt Controller Type Register.....	72
5.11 IMPLEMENTATION DEFINED registers summary.....	73
5.12 Direct cache access registers.....	76
5.12.1 DCAICLR and DCADCLR, Direct Cache Access Location Registers.....	76
5.12.2 DCAICRR and DCADCRR, Direct Cache Access Read Registers.....	79
5.13 Error bank registers.....	84
5.13.1 IEBR0 and IEBR1, Instruction Cache Error Bank Register 0-1.....	85
5.13.2 DEBR0 and DEBR1, Data Cache Error Bank Register 0-1.....	86
5.13.3 TEBR0 and TEBR1, TCM Error Bank Register 0-1.....	88
5.14 MSCR, Memory System Control Register.....	90
5.15 PAHBCR, P-AHB Control Register.....	93
5.16 Power mode control registers.....	94
5.16.1 CPDLPSTATE, Core Power Domain Low Power State Register.....	95
5.16.2 DPDLPSTATE, Debug Power Domain Low Power State Register.....	96
5.17 Processor configuration information registers.....	97
5.17.1 CFGINFOSEL, Processor configuration information selection register.....	98
5.17.2 CFGINFORD, Processor configuration information read data register.....	101
5.18 ID_PFR0, Processor Feature Register 0.....	103

5.19 ITCMCR and DTCMCR, TCM Control Registers.....	104
5.20 TCM security gate registers.....	105
5.20.1 ITGU_CTRL and DTGU_CTRL, ITGU and DTGU Control Registers.....	106
5.20.2 ITGU_CFG and DTGU_CFG, ITGU and DTGU Configuration Registers.....	107
5.20.3 ITGU_LUTn and DTGU_LUTn, ITGU and DTGU Look Up Table Registers.....	108
5.21 EWIC interrupt status access registers.....	110
5.21.1 EVENTSPR, Event Set Pending Register.....	111
5.21.2 EVENTMASKA and EVENTMASKn, n=0-14, Wakeup Event Mask Registers.....	112
5.22 STL observation registers.....	113
5.22.1 STLNVICPENDOR and STLNVICACTVOR, NVIC observation registers.....	114
5.22.2 STLIDMPUSR, STLIMPUOR, and STLDMPUOR, MPU observation registers.....	115
<b>6. Initialization.....</b>	<b>118</b>
6.1 Initialization overview.....	118
6.2 Initializing and reprogramming the MPU.....	118
6.3 Initializing the EPU.....	119
6.4 Programming the SAU.....	119
6.5 Initializing the instruction and data cache.....	120
6.5.1 Enabling the instruction and data cache.....	121
6.5.2 Powering down the caches.....	121
6.5.3 Powering up the caches.....	122
6.6 Enabling the branch cache.....	122
6.7 Enabling and preloading the TCM.....	123
6.8 Enabling and locking the TCM security gates.....	124
6.9 Enabling the P-AHB interface.....	124
<b>7. Power management.....</b>	<b>125</b>
7.1 Power domains.....	125
7.2 Power states.....	126
7.3 Power and operating mode transitions.....	127
7.3.1 Operating mode transitions which change PDRAMS power state.....	129
7.4 Core P-Channel and power mode selection.....	130
7.4.1 P-Channel interface tie-off when P-Channel is not used.....	131
7.5 COREPACTIVE and required power mode.....	131
7.5.1 COREPACTIVE signal encoding.....	133
7.6 PDCORE low-power requirements.....	133
7.7 PDRAMS powerdown requirements.....	134

7.8 Warm reset power mode.....	135
7.9 Debug Q-Channel and PDDEBUG power domain.....	136
7.10 Q-Channel clock control.....	137
7.11 PWRDBGWAKEQACTIVE.....	138
<b>8. Memory model.....</b>	<b>139</b>
8.1 Memory map.....	139
8.2 Memory types.....	140
8.3 Private Peripheral Bus.....	141
8.4 Unaligned accesses.....	144
8.5 Access privilege level for Device and Normal memory.....	145
8.6 Memory ordering and barriers.....	146
8.7 Execute Only Memory.....	146
<b>9. Memory Authentication.....</b>	<b>148</b>
9.1 MAU features.....	148
9.2 Security Attribution Unit.....	148
9.2.1 SAU register summary.....	149
9.2.2 Security levels.....	150
9.3 Memory Protection Unit.....	150
9.3.1 Memory Protection Unit register summary.....	151
9.4 Implementation Defined Attribution Unit.....	152
9.4.1 IDAU interface and backwards compatibility.....	152
9.5 Security attribution signals.....	153
9.6 TCM Gate Units.....	154
9.7 TCM and P-AHB security access control.....	154
9.7.1 Memory aliasing and IDAU/SAU configuration.....	155
9.7.2 Security access gating using the TGU.....	159
9.7.3 TGU configuration.....	159
9.7.4 Security check and fault response.....	161
<b>10. Memory system.....</b>	<b>162</b>
10.1 Memory system features.....	162
10.2 Memory system faults.....	164
10.2.1 Classes of fault.....	165
10.3 Memory system behavior.....	167
10.3.1 Speculative accesses.....	169



10.3.2 Access privilege level for Device and Normal memory.....	170
10.4 M-AXI interface.....	171
10.4.1 High performance M-AXI configuration.....	172
10.4.2 Area optimized M-AXI configuration.....	174
10.4.3 Bridging to AHB.....	175
10.4.4 Write response.....	176
10.4.5 Memory system implications for AXI accesses.....	176
10.4.6 M-AXI interface transfers.....	177
10.5 AHB Main (M-AHB) interface.....	179
10.6 Peripheral AHB interface.....	180
10.6.1 P-AHB interface transfers.....	180
10.6.2 P-AHB interface configuration.....	181
10.6.3 P-AHB considerations.....	182
10.7 TCM-AHB interface.....	183
10.7.1 TCM-AHB memory map.....	184
10.7.2 TCM-AHB transfers.....	185
10.7.3 TCM-AHB interface arbitration.....	187
10.7.4 TCM-AHB availability and low power states.....	187
10.8 EPPB interface.....	188
10.9 TCM interfaces.....	189
10.9.1 TCM configuration.....	190
10.9.2 TCM transactions.....	191
10.9.3 Booting from TCM.....	192
10.9.4 Integration with flash memory.....	192
10.9.5 System access to TCM through the TCM-AHB DMA interface.....	193
10.10 Instruction cache, data cache, and unified cache.....	194
10.10.1 L1 data cache.....	196
10.10.2 L1 instruction cache.....	197
10.10.3 L1 unified cache.....	198
10.10.4 Cache maintenance operations.....	199
10.10.5 Automatic cache invalidation at reset.....	200
10.10.6 Cache coherency.....	201
10.10.7 Accessing the caches.....	202
10.10.8 System cache support.....	203
10.10.9 Direct cache access.....	203
10.11 Store buffer.....	206

10.11.1 Store buffer merging.....	206
10.11.2 Store buffer behavior.....	207
10.11.3 Store buffer ordering.....	207
10.11.4 Store buffer draining.....	207
10.12 Internal local exclusive access monitor.....	208
10.13 Main interface and P-AHB interaction with the global exclusive monitor.....	209
10.14 MBIST.....	209
<b>11. Reliability, Availability, and Serviceability Extension support.....</b>	<b>211</b>
11.1 Cortex®-M52 processor implementation of RAS.....	211
11.1.1 Cortex®-M52 RAS events.....	212
11.2 ECC memory protection behavior.....	212
11.2.1 ECC schemes and error type terminology.....	213
11.2.2 Enabling ECC.....	214
11.2.3 Error detection and processing.....	215
11.2.4 Error reporting.....	218
11.2.5 Address decoder protection and white noise protection.....	221
11.3 Flop parity.....	221
11.4 Interface protection behavior.....	222
11.5 RAS memory barriers.....	225
11.6 RAS Extension registers.....	225
11.6.1 ERRFR0, RAS Error Record Feature Register.....	226
11.6.2 ERRSTATUS0, RAS Error Record Primary Status Register.....	227
11.6.3 ERRADDR0 and ERRADDR20, RAS Error Record Address Registers.....	230
11.6.4 ERRMISC10, Error Record Miscellaneous Register 10.....	231
11.6.5 ERRGSR0, RAS Fault Group Status Register.....	232
11.6.6 ERRDEVID, RAS Error Record Device ID Register.....	233
11.6.7 RFSR, RAS Fault Status Register.....	234
<b>12. Nested Vectored Interrupt Controller.....</b>	<b>236</b>
12.1 NVIC features.....	236
12.2 Registers associated with interrupt control and behavior.....	236
12.3 NVIC register summary.....	237
12.4 Software Interrupt Generation register summary.....	238
12.5 SysTick Timer register summary.....	238
<b>13. External coprocessors.....</b>	<b>239</b>

13.1 External coprocessors features.....	239
13.2 Operation.....	239
13.3 Data transfer rates.....	240
13.4 Coprocessor instruction restrictions.....	240
13.5 Debug access to coprocessor registers usage constraints.....	241
13.6 Exceptions and context switch.....	241
13.7 Response to coprocessor errors.....	241
13.8 Hazard between load and store instructions followed by coprocessor transactions.....	242
<b>14. Arm Custom Instructions.....</b>	<b>243</b>
14.1 Arm Custom Instructions support.....	243
14.2 Usage restrictions.....	245
<b>15. Floating-point and MVE support.....</b>	<b>246</b>
15.1 Floating-point and MVE operation.....	246
15.1.1 EPU views of the register bank.....	247
15.1.2 Modes of operation.....	247
15.1.3 Compliance with the IEEE 754 standard.....	247
15.1.4 Exceptions.....	248
15.2 Floating-point and MVE register summary.....	248
15.3 FPDSCR and FPSCR register reset values.....	249
<b>16. Debug.....</b>	<b>250</b>
16.1 Debug functionality.....	250
16.1.1 CoreSight™ discovery.....	251
16.1.2 Debugger actions for identifying the processor.....	253
16.1.3 Processor ROM table identification and entries.....	253
16.1.4 Debug identification block register summary.....	255
16.1.5 Debug register summary.....	256
16.2 D-AHB interface.....	257
16.2.1 Debug memory access.....	257
16.2.2 Debugger access memory attributes and data cache access.....	259
16.2.3 Debug access security and attributes.....	261
16.2.4 Debug during reset and before code execution commences.....	262
16.2.5 Advanced DSP debug capabilities.....	264
<b>17. Performance Monitoring Unit Extension.....</b>	<b>265</b>

17.1 PMU features.....	265
17.2 PMU events.....	266
17.3 PMU register summary.....	271
<b>18. Instrumentation Trace Macrocell.....</b>	<b>273</b>
18.1 ITM features.....	273
18.2 ITM register summary.....	274
18.3 ITM_TPR, ITM Trace Privilege Register.....	276
18.4 ITM_ITCTRL, ITM Integration Mode Control Register.....	276
18.5 ITM_ITWRITE, Integration Write Register.....	277
18.6 ITM_ITREAD, Integration Read Register.....	278
<b>19. Data Watchpoint and Trace unit.....</b>	<b>280</b>
19.1 DWT features.....	280
19.2 DWT debug access control.....	281
19.3 DWT comparators.....	283
19.4 Cycle counter and profiling counters.....	284
19.5 DWT register summary.....	285
<b>20. Cross Trigger Interface.....</b>	<b>288</b>
20.1 CTI features.....	288
20.2 CTI register summary.....	290
20.3 CTI_CONTROL, CTI Control Register.....	291
20.4 CTI_INACK, CTI Interrupt Acknowledge Register.....	292
20.5 CTI_APPSET, CTI Application Channel Set Register.....	293
20.6 CTI_APPCLR, CTI Application Channel Clear Register.....	294
20.7 CTI_APPPULSE, CTI Application Channel Pulse Register.....	295
20.8 CTI_INEN<n>, n=0-5, CTI Trigger <n> to Channel Enable Register.....	295
20.9 CTI_OUTEN<n>, n=0-7, CTI Channel <n> to Trigger Enable Register.....	297
20.10 CTI_TRIGINSTATUS, CTI Trigger Input Status Register.....	298
20.11 CTI_TRIGOUTSTATUS, CTI Trigger Output Status Register.....	299
20.12 CTI_CHINSTATUS, CTI Channel Input Status Register.....	300
20.13 CTI_CHOUTSTATUS, CTI Channel Output Status Register.....	301
20.14 CTI_CHANNELGATE, CTI Channel Gate Register.....	301
20.15 CTI_ITCHOUT, Integration Test Channel Output Register.....	302
20.16 CTI_ITTRIGOUT, Integration Test Trigger Output Register.....	303
20.17 CTI_ITCHIN, Integration Test Channel Input Register.....	304

20.18 CTI_ITTRIGIN, Integration Test Trigger Input Register.....	305
20.19 CTI_ITCONTROL, Integration Mode Control Register.....	306
20.20 CTI_DEVARCH, Device Architecture Register.....	307
20.21 CTI_DEVID, Device Configuration Register.....	308
20.22 CTI_DEVTYPE, Device Type Identifier Register.....	309
20.23 CTI_PIDR4, Peripheral Identification Register 4.....	310
20.24 CTI_PIDR5, Peripheral Identification Register 5.....	310
20.25 CTI_PIDR6, Peripheral Identification Register 6.....	311
20.26 CTI_PIDR7, Peripheral Identification Register 7.....	312
20.27 CTI_PIDR0, Peripheral Identification Register 0.....	313
20.28 CTI_PIDR1, Peripheral Identification Register 1.....	313
20.29 CTI_PIDR2, Peripheral Identification Register 2.....	314
20.30 CTI_PIDR3, Peripheral Identification Register 3.....	315
20.31 CTI_CIDR0, Component Identification Register 0.....	316
20.32 CTI_CIDR1, Component Identification Register 1.....	317
20.33 CTI_CIDR2, Component Identification Register 2.....	317
20.34 CTI_CIDR3, Component Identification Register 3.....	318

## **21. BreakPoint Unit.....320**

21.1 BPU features.....	320
21.2 BPU register summary.....	320

## **A. External Wakeup Interrupt Controller.....322**

A.1 EWIC features.....	322
A.2 EWIC register summary.....	323
A.2.1 EWIC_CR, EWIC Control Register.....	323
A.2.2 EWIC_ASCR, EWIC Automatic Sequence Control Register.....	324
A.2.3 EWIC_CLRMASK, EWIC Clear Mask Register.....	326
A.2.4 EWIC_NUMID, EWIC Event Number ID Register.....	326
A.2.5 EWIC_MASKA and EWIC_MASKn, EWIC Mask Registers.....	327
A.2.6 EWIC_PENDA and EWIC_PENDn, EWIC Pend Event Registers.....	328
A.2.7 EWIC_PSR, EWIC Pend Summary Register.....	330
A.2.8 EWIC CoreSight™ register summary.....	331
A.2.9 EWIC_CLAIMSET, EWIC Claim Tag Set Register.....	332
A.2.10 EWIC_CLAIMCLR, EWIC Claim Tag Clear Register.....	333

## **B. Trace Port Interface Unit.....335**

B.1 TPIU features.....	335
B.1.1 TPIU Formatter.....	336
B.1.2 Serial Wire Output format.....	337
B.2 TPIU register summary.....	337
B.2.1 TPIU_SSPSR, Supported Port Size Register.....	339
B.2.2 TPIU_CSPSR, Current Port Size Register.....	340
B.2.3 TPIU_SPPR, Selected Pin Protocol Register.....	341
B.2.4 TPIU_PSCR, Periodic Synchronization Counter Register.....	342
B.2.5 TPIU_ACPR, Asynchronous Clock Prescaler Register.....	342
B.2.6 TPIU_FFSR, Formatter and Flush Status Register.....	343
B.2.7 TPIU_FFCR, Formatter and Flush Control Register.....	344
B.2.8 TPIU_TRIGGER, TPIU TRIGGER Register.....	345
B.2.9 ITFTTD0, Integration Test FIFO Test Data 0 Register.....	346
B.2.10 ITATBCTR2, Integration Test ATB Control Register 2.....	347
B.2.11 ITFTTD1, Integration Test FIFO Test Data 1 Register.....	347
B.2.12 ITATBCTR0, Integration Test ATB Control 0 Register.....	348
B.2.13 TPIU_ITCTRL, Integration Mode Control.....	349
B.2.14 CLAIMSET, Claim Tag Set Register.....	350
B.2.15 CLAIMCLR, Claim Tag Clear Register.....	351
B.2.16 TPIU_DEVID, Device Configuration Register.....	352
B.2.17 TPIU_DEVTYPE, Device Type Identifier Register.....	353
B.2.18 TPIU_PIDR4, Peripheral Identification Register 4.....	353
B.2.19 TPIU_PIDR5, Peripheral Identification Register 5.....	354
B.2.20 TPIU_PIDR6, Peripheral Identification Register 6.....	355
B.2.21 TPIU_PIDR7, Peripheral Identification Register 7.....	356
B.2.22 TPIU_PIDR0, Peripheral Identification Register 0.....	356
B.2.23 TPIU_PIDR1, Peripheral Identification Register 1.....	357
B.2.24 TPIU_PIDR2, Peripheral Identification Register 2.....	358
B.2.25 TPIU_PIDR3, Peripheral Identification Register 3.....	359
B.2.26 TPIU_CIDR0, Component Identification Register 0.....	360
B.2.27 TPIU_CIDR1, Component Identification Register 1.....	360
B.2.28 TPIU_CIDR2, Component Identification Register 2.....	361
B.2.29 TPIU_CIDR3, Component Identification Register 3.....	362
<b>C. Signal Descriptions.....</b>	<b>363</b>
C.1 Clock and clock enable signals.....	363

C.2 Reset signals.....	363
C.3 Static configuration signals.....	364
C.4 Reset configuration signals.....	366
C.5 Cache initialization signal.....	367
C.6 Instruction execution control signals.....	368
C.7 Instruction Tightly Coupled Memory interface signals.....	369
C.8 Data Tightly Coupled Memory interface signals.....	370
C.9 M-AXI interface signals.....	372
C.9.1 M-AXI interface protection signals.....	374
C.10 CODE-AHB interface signals.....	376
C.10.1 CODE-AHB interface protection signals.....	377
C.11 SYS-AHB interface signals.....	377
C.11.1 SYS-AHB interface protection signals.....	378
C.12 TCM-AHB interface signals.....	379
C.12.1 TCM-AHB interface protection signals.....	380
C.13 P-AHB interface signals.....	380
C.13.1 P-AHB interface protection signals.....	381
C.14 D-AHB interface signals.....	382
C.14.1 D-AHB interface protection signals.....	382
C.15 EPPB interface signals.....	383
C.15.1 EPPB interface protection signals.....	383
C.16 External coprocessor interface signals.....	384
C.17 Arm Custom Instructions signals.....	385
C.18 Debug interface signals.....	385
C.19 P-Channel and Q-Channel power control signals.....	386
C.20 Q-Channel clock control signals.....	387
C.21 Power compatibility control signals.....	388
C.22 ITM interface signals.....	388
C.23 ETM interface signals.....	388
C.24 Trace synchronization and trigger signals.....	389
C.25 CTI interface signals.....	389
C.26 Interrupt signals.....	389
C.27 WIC interface signals.....	390
C.28 Event signals.....	392
C.29 IDAU interface signals.....	392
C.30 Miscellaneous signals.....	393

C.31 Error interface signals.....	397
C.32 Floating-point exception signals.....	398
C.33 PMC-100 interface signals.....	398
C.33.1 PMC-100 APB interface protection signals.....	400
C.34 Test interface signals.....	400
C.35 DCLS operation signals.....	401
C.35.1 Control and reporting.....	401
<b>D. UNPREDICTABLE Behaviors.....</b>	<b>405</b>
D.1 Use of instructions defined in architecture variants.....	405
D.2 Use of Program Counter - R15 encoding.....	405
D.3 Use of Stack Pointer - as a general-purpose register R13.....	405
D.4 Register list in load and store multiple instructions.....	406
D.5 Exception-continuable instructions.....	406
D.6 Stack limit checking.....	407
D.7 UNPREDICTABLE instructions within an IT block.....	407
D.8 Memory access and address space.....	408
D.9 MPU programming.....	409
D.10 Miscellaneous UNPREDICTABLE instruction behavior.....	410
<b>E. Revisions.....</b>	<b>411</b>
E.1 Revisions.....	411



# 1. Introduction

## 1.1 Product revision status

The  $r_xp_y$  identifier indicates the revision status of the product described in this manual, for example,  $r1p2$ , where:

<b><math>r_x</math></b>	Identifies the major revision of the product, for example, $r1$ .
<b><math>p_y</math></b>	Identifies the minor revision or modification status of the product, for example, $p2$ .

## 1.2 Intended audience

This manual is written to help system designers, system integrators, verification engineers, and software programmers who are implementing a System on Chip (SoC) device based on the Cortex®-M52 processor.

## 1.3 Conventions

The following subsections describe conventions used in Arm documents.

### Glossary

The Arm® Glossary is a list of terms used in Arm documentation, together with definitions for those terms. The Arm Glossary does not contain terms that are industry standard unless the Arm meaning differs from the generally accepted meaning.

See the Arm Glossary for more information: [developer.arm.com/glossary](https://developer.arm.com/glossary).

Convention	Use
<i>italic</i>	Citations.
<b>bold</b>	Terms in descriptive lists, where appropriate.
monospace	Text that you can enter at the keyboard, such as commands, file and program names, and source code.
monospace <u>underline</u>	A permitted abbreviation for a command or option. You can enter the underlined text instead of the full command or option name.

Convention	Use
<and>	Encloses replaceable terms for assembler syntax where they appear in code or code fragments.  For example:  <pre>MRC p15, 0, &lt;Rd&gt;, &lt;CRn&gt;, &lt;CRm&gt;, &lt;Opcode_2&gt;</pre>
SMALL CAPITALS	Terms that have specific technical meanings as defined in the <i>Arm® Glossary</i> . For example, <b>IMPLEMENTATION DEFINED</b> , <b>IMPLEMENTATION SPECIFIC</b> , <b>UNKNOWN</b> , and <b>UNPREDICTABLE</b> .



Recommendations. Not following these recommendations might lead to system failure or damage.



Requirements for the system. Not following these requirements might result in system failure or damage.



Requirements for the system. Not following these requirements will result in system failure or damage.



An important piece of information that needs your attention.



A useful tip that might make it easier, better or faster to perform a task.



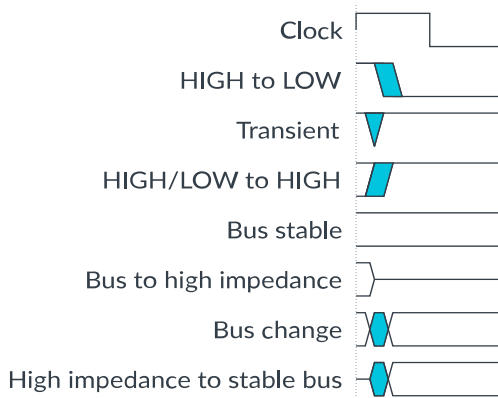
A reminder of something important that relates to the information you are reading.

## Timing diagrams

The following figure explains the components used in timing diagrams. Variations, when they occur, have clear labels. You must not assume any timing information that is not explicit in the diagrams.

Shaded bus and signal areas are undefined, so the bus or signal can assume any value within the shaded area at that time. The actual level is unimportant and does not affect normal operation.

**Figure 1-1: Key to timing diagram conventions**



## Signals

The signal conventions are:

### Signal level

The level of an asserted signal depends on whether the signal is active-HIGH or active-LOW. Asserted means:

- HIGH for active-HIGH signals.
- LOW for active-LOW signals.

### Lowercase n

At the start or end of a signal name, n denotes an active-LOW signal.

## 1.4 Useful resources

This document contains information that is specific to this product. See the following resources for other useful information.

Access to Arm documents depends on their confidentiality:

- Non-Confidential documents are available at [developer.arm.com/documentation](https://developer.arm.com/documentation). Each document link in the following tables goes to the online version of the document.
- Confidential documents are available to licensees only through the product package.

Arm and Arm China product resources	Document ID	Confidentiality
<i>Arm China CoreSight™ ETM-M52 Technical Reference Manual</i>	102774	Non-Confidential
<i>Arm China Cortex®-M52 Processor Integration and Implementation Manual</i>	102775	Confidential
<i>Arm® CoreSight™ TPIU-M Technical Reference Manual</i>	102427	Non-Confidential
<i>Arm® PMC-100 Technical Reference Manual</i>	101528	Non-Confidential

Arm and Arm China product resources	Document ID	Confidentiality
Arm®v8.1-M Performance Monitoring User Guide Application Note	ARM051-799564642-251	Non-Confidential

Arm architecture and specifications	Document ID	Confidentiality
AMBA® 4 ATB Protocol Specification	IHI 0032	Non-Confidential
AMBA® APB Protocol Version 2.0 Specification	IHI 0024	Non-Confidential
AMBA® AXI and ACE Protocol Specification	IHI 0022	Non-Confidential
AMBA® Low Power Interface Specification	IHI 0068	Non-Confidential
Arm® AMBA® 5 AHB Protocol Specification	IHI 0033	Non-Confidential
Arm® CoreSight™ Architecture Specification v3.0	IHI 0029	Non-Confidential
Arm® Debug Interface Architecture Specification, ADIv6.0	IHI 0074	Non-Confidential
Arm® Embedded Trace Macrocell Architecture Specification ETMv4	IHI 0064	Non-Confidential
Arm® Reliability, Availability, and Serviceability (RAS) Specification	DDI 0587	Non-Confidential
Arm®v8-M Architecture Reference Manual	DDI 0553	Non-Confidential

Non-Arm resources	Document ID	Organization
IEEE Standard for Binary Floating-Point Arithmetic	ANSI/IEEE Std 754-2008	<a href="https://www.ieee.org">https://www.ieee.org</a>
Test Access Port and Boundary-Scan Architecture (JTAG)	IEEE Std 1149.1-2001	<a href="https://www.ieee.org">https://www.ieee.org</a>



Note

Arm tests its PDFs only in Adobe Acrobat and Acrobat Reader. Arm cannot guarantee the quality of its documents when used with any other PDF reader.

Adobe PDF reader products can be downloaded at <http://www.adobe.com>.

## 2. The Cortex®-M52 processor

This chapter provides an overview of the Cortex®-M52 processor and its features.

### 2.1 Cortex®-M52 processor overview

The Cortex®-M52 processor is a fully synthesizable mid-range microcontroller class processor that implements the Arm®v8.1-M Mainline architecture which includes support for the *M-profile Vector Extension* (MVE) and *Pointer Authentication and Branch Target Identification* (PACBTI) Extension. The processor also supports previous Arm®v8-M architectural features.

The design is focused on compute applications such as *Digital Signal Processing* (DSP) and machine learning. The Cortex®-M52 processor is energy efficient and achieves high compute performance across scalar and vector operations while maintaining low power consumption.

The processor can be configured to include *Dual Core Lockstep* (DCLS) functionality, which implements a redundant copy of most of the processor logic.

To support *Arm Custom Instructions* (ACIs), the processor includes optional *Custom Datapath Extension* (CDE) modules, which are embedded inside the logic. These modules are used to execute user-defined instructions that work on general-purpose integer, floating point, and MVE registers.

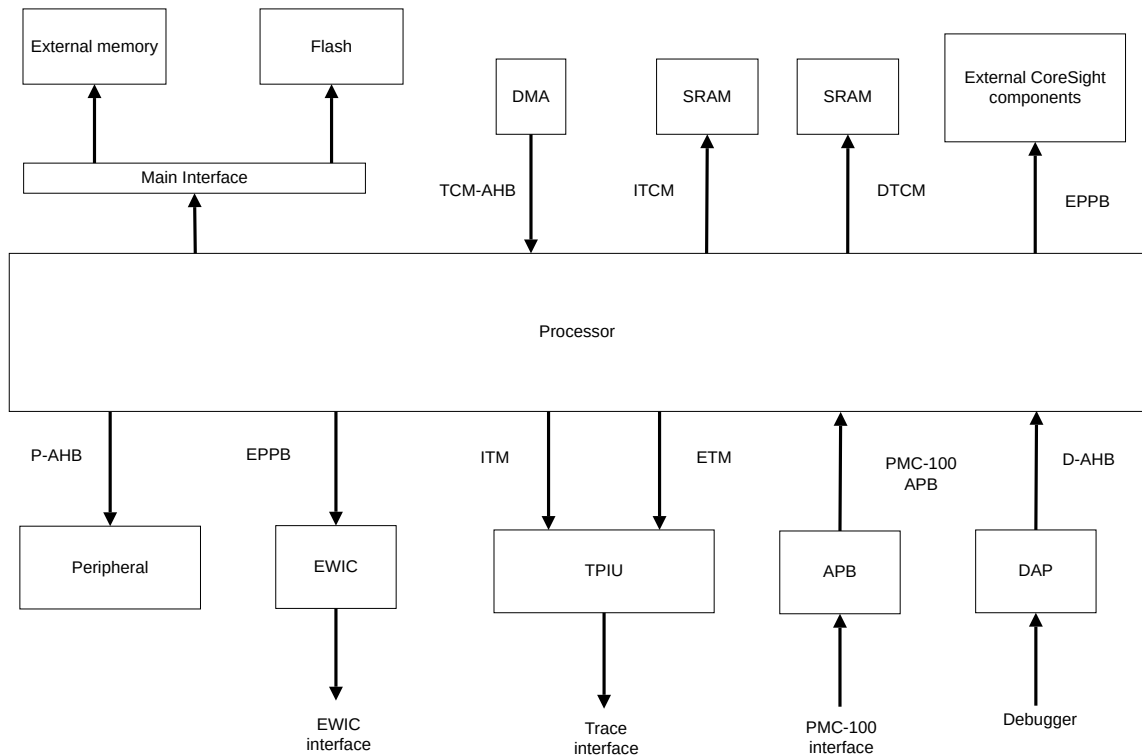


Where CDE is mentioned in this document, it is referring to the support of *Arm Custom Instructions* (ACIs).

---

The following figure shows the Cortex®-M52 processor in a typical system.

**Figure 2-1: Example processor system**



For more information on the processor-level components, see [Cortex-M52 processor components](#).

## 2.2 Cortex®-M52 features

The Cortex®-M52 processor implements the Arm®v8.1-M Mainline architecture and also supports previous Arm®v8-M architectural features.

For more information on Arm®v8.1-M and Arm®v8-M features and variants information, see the *Arm®v8-M variants* section in the *Arm®v8-M Architecture Reference Manual*.



- The 'Optional' column indicates a feature that can be optionally included, either by:
  - Setting relevant RTL parameters. For example, if you include the *Instrumentation Trace Macrocell* (ITM).
  - Being optionally licensed. For example, if you optionally license Security Extension.

- The 'Configurable' column indicates a feature that can be configured to any permitted value by setting relevant RTL parameters. For example, you can configure the size of the instruction and data cache to be 4KB, 8KB, 16KB, 32KB, or 64KB.

**Table 2-1: Cortex®-M52 processor architectural features**

Feature	Architecture version	Always present?	Optional?	Configurable?	Details
Arm PMSAv8 memory system architecture with memory protection	-	Yes	-	-	-
Arm FpV5 hardware supporting scalar half, single, and double-precision floating-point operation that is compliant with IEEE754-2008	Arm®v8-M onwards	-	Yes	Yes	Can be configured to support the following operations: <ul style="list-style-type: none"> <li>Scalar half- and single-precision operation</li> <li>Scalar half-, single-, and double-precision floating-point operation</li> </ul>
DSP Extension	Arm®v8-M onwards	Yes	-	-	-
DSP Debug Extension	Arm®v8.1-M	Yes	-	-	-
Exception model	Arm®v8-M onwards	Yes	-	-	See <a href="#">Exceptions</a> for more information.
External <i>Implementation Defined Attribution Unit</i> (IDAU)	-	Yes	-	-	Can be used only when the Security Extension is enabled
Level 1 (L1) unified cache, instruction cache, and data cache.	Arm®v8-M onwards	-	Yes	Yes	-
PACBTI Extension	Arm®v8.1-M	-	Yes	-	-
Main Extension	Arm®v8.1-M	Yes	-	-	Includes the 16-bit and 32-bit Thumb instruction set
Memory Protection Unit (MPU)	Arm®v8-M onwards	-	Yes	Yes	Supports up to 16 regions each for Secure and Non-secure applications
MVE, supporting <i>Single Instruction Multiple Data</i> (SIMD) 128-bit vector operations	Arm®v8.1-M	-	Yes	-	Supported data types are: <ul style="list-style-type: none"> <li>Integer</li> <li>Half precision floating-point (supported when floating-point functionality is included)</li> <li>Single precision floating-point (supported when floating-point functionality is included)</li> </ul> MVE is also referred to as Arm® Helium® technology.
Low Overhead Branch (LOB) Extension	Arm®v8.1-M	Yes	-	-	Low overhead loop and branch info cache are supported. Branch Future instruction is treated as NOP. See <i>Arm®v8-M Architecture Reference Manual</i> .
Support for <i>Data Independent Timing</i> (DIT) operation	Arm®v8.1-M	Yes	-	-	See <i>Arm®v8-M Architecture Reference Manual</i> .

Feature	Architecture version	Always present?	Optional?	Configurable?	Details
<i>Nested Vector Interrupt Controller (NVIC)</i>	Arm®v8-M onwards	Yes	-	Yes	Supports up to 480 external interrupts with up to 256 priority levels
<i>Reliability, Availability, and Serviceability (RAS) Extension</i>	Arm®v8.1-M	Yes	-	-	-
<i>Security Attribution Unit (SAU)</i>	Arm®v8-M onwards	-	Yes	Yes	Supports up to eight Non-secure or Non-secure Callable memory regions
<i>Security Extension</i>	Arm®v8-M onwards	-	Yes	-	The Security Extension is an implementation of Arm® TrustZone®
<i>Unprivileged Debug Extension (UDE)</i>	Arm®v8.1-M	Yes	-	-	-
<i>Custom Datapath Extension (CDE)</i>	Arm®v8-M	-	Yes	Yes	Support for CDE adds classes of <i>Arm Custom Instructions (ACIs)</i> in the coprocessor instruction space.

## Debug and trace features

The following table shows the debug and trace features of the processor.

**Table 2-2: Debug and trace features**

Feature	Architecture version	Always present?	Optional?	Configurable?	Details
<i>BreakPoint Unit (BPU) and comparator support</i>	Arm®v8-M onwards	-	Yes	Yes	Four or eight comparators are supported
<i>Data Watchpoint and Trace (DWT) unit and comparator support</i>	Arm®v8-M onwards	-	Yes	Yes	Support the <i>Performance Monitoring Unit (PMU)</i> . Two, four, or eight comparators are supported.
<i>Embedded Trace Macrocell (ETM)</i>	Arm® (ETM) v4.5	-	Yes	-	Set relevant RTL parameters.
ITM	Arm®v8-M onwards	-	Yes	-	-
PMU	Arm®v8.1-M	-	Yes	-	Present when the DWT is included.

## 2.3 Supported standards and specifications

The Cortex®-M52 processor complies with, or implements, the relevant Arm® architectural standards and protocols.

This book complements architecture reference manuals, architecture specifications, protocol specifications, and relevant external standards. It does not duplicate information from these sources.

### Arm® architecture

The Cortex®-M52 processor is compliant with the Arm®v8.1-M Mainline architecture and also supports previous Arm®v8-M architectural features. See [Cortex-M52 features](#) for more information.



## Bus architecture

The Cortex®-M52 processor implements either AMBA® 5 AXI-compliant AXI Main (M-AXI) interface or AMBA® 5 AHB-compliant AHB Main (M-AHB) interface (configurable) for slow on-chip or off-chip memory and devices.

It also provides external interfaces that comply with the AMBA® 5 AHB protocol.

Additionally, the Cortex®-M52 processor implements interfaces for CoreSight™ and other debug components and optional PMC-100 controller for on-line MBIST using the AMBA® 4 APB protocol (this is the same as APB protocol version 2.0) and the ATBv1.1 part of the AMBA® 4 ATB protocol.

For more information, see the:

- *AMBA® AXI and ACE Protocol Specification*
- *Arm® AMBA® 5 AHB Protocol Specification*
- *AMBA® APB Protocol Version 2.0 Specification*
- *AMBA® 4 ATB Protocol Specification*

The Cortex®-M52 processor also provides P-Channel and Q-Channel interfaces for power and clock control. See the *AMBA® Low Power Interface Specification Arm® Q-Channel and P-Channel Interfaces*.

For more overview information on bus interfaces, see [Interfaces](#).

## Debug

The debug features of the Cortex®-M52 processor implement the Arm® Debug Interface v6.0 architecture.

See the *Arm® Debug Interface Architecture Specification, ADIv6.0*.

## Embedded Trace Macrocell

The trace features of the Cortex®-M52 processor implement the Arm® *Embedded Trace Macrocell* (ETM) v4.5 architecture.

See the *Arm China CoreSight™ ETM-M52 Technical Reference Manual* for more information.

## Extension Processing Unit

The *Extension Processing Unit* (EPU) performs scalar floating-point and vector operations.

The EPU is configured to include a scalar floating-point functionality, which supports either half-precision, single-precision arithmetic or half-precision, single-precision, and double-precision arithmetic (configurable) as defined by the Arm® FPUv5 architecture.

The EPU implements MVE, which can support:

- Half-precision, single-precision floating-point, or half-precision, single-precision, and double-precision floating-point (configurable)

- Integer, half-precision, and single-precision vector arithmetic

See [Cortex-M52 implementation options](#).

The Cortex®-M52 processor provides floating-point computation functionality that is included with Floating-point and MVE, which is compliant with the *ANSI/IEEE Std 754-2008, IEEE Standard for Binary Floating-Point Arithmetic*.

## 2.4 Design tasks

The Cortex®-M52 processor is delivered as synthesizable RTL that must go through implementation, integration, and programming processes before you can use it in a product.

The following definitions describe each top-level process in the design flow:

### Implementation

The implementer configures and synthesizes the RTL.

### Integration

The integrator connects the Cortex®-M52 processor into an SoC. This includes connecting it to a memory system and peripherals.

### Programming

The system programmer develops the software required to configure and initialize the Cortex®-M52 processor and tests the required application software.

Implementation and integration choices affect the behavior and features of the Cortex®-M52 processor.

The operation of the final device depends on:

### Build configuration

The implementer chooses the options that affect how the RTL source files are pre-processed. These options usually include or exclude logic that affects one or more of the area, maximum frequency, and features of the resulting macrocell.

### Configuration inputs

The integrator configures some features of the Cortex®-M52 processor by tying inputs to specific values. These configurations affect the start-up behavior before any software configuration is made. They can also limit the options available to the software.

### Software configuration

The programmer configures the Cortex®-M52 processor by programming particular values into registers. This affects the behavior of the Cortex®-M52 processor.



This manual refers to **IMPLEMENTATION-DEFINED** features that are applicable to build configuration options. Reference to a feature that is included means that the

appropriate build and signal configuration options have been selected. Reference to an enabled feature means that software has also configured the feature.

---

## 2.5 Documentation

The Cortex®-M52 processor documentation can help you complete the top-level processes of implementation, integration, and programming that are required to use the product correctly.

The Cortex®-M52 processor documentation includes a Technical Reference Manual, an Integration and Implementation Manual, and User Guide Reference Material.

### Technical Reference Manual

The *Technical Reference Manual* (TRM) describes the functionality and the effects of functional options on the behavior of the Cortex®-M52 processor. It is required at all stages of the design flow. Some behavior described in the TRM might not be relevant because of the way that the Cortex®-M52 processor is implemented and integrated. If you are programming the Cortex®-M52 processor, then contact the implementer to determine:

- The build configuration of the implementation.
- What integration, if any, was performed before implementing the Cortex®-M52 processor.

### Integration and Implementation Manual

The *Integration and Implementation Manual* (IIM) describes:

- The available build configuration options and related issues in selecting them.
- How to configure the *Register Transfer Level* (RTL) with the build configuration options.
- How to integrate the Cortex®-M52 processor into an SoC. This book includes a description of the integration kit and describes the pins that the integrator must tie off to configure the macrocell for the required integration.
- How to implement the Cortex®-M52 processor into your design. This book includes *Memory Built-In Self Test* (MBIST) and *Design for Test* (DFT) information, and information on how to perform netlist dynamic verification on the Cortex®-M52 processor.
- The processes to sign off the integration and implementation of the design.

The Arm China product deliverables include reference scripts and information about using them to implement your design.

Reference methodology documentation from your EDA tools vendor and the *implementation Reference Methodology* (IRM) `readme.txt` provided by Arm China complements the IIM.

The IIM is a confidential book that is only available to licensees and partners with an NDA agreement.

### User Guide Reference Material

This document provides reference material that the partners can configure and include in a User Guide for a Cortex®-M52 processor. Typically:

- Each chapter in this reference material might correspond to a section in the User Guide.
- Each top-level section in this reference material might correspond to a chapter in the User Guide.

However, you can organize this material in any way, subject to the conditions of the license agreement under which the material is supplied.

See the [Useful resources](#) for more information about the books that are associated with the Cortex®-M52 processor.

## 2.6 Product revisions

The following product revisions have been released.

<b>r0p0</b>	First beta release for r0p0; first limited access release for r0p0
<b>r0p1</b>	First early access release for r0p1; second early access release for r0p1
<b>r0p2</b>	First release for r0p2

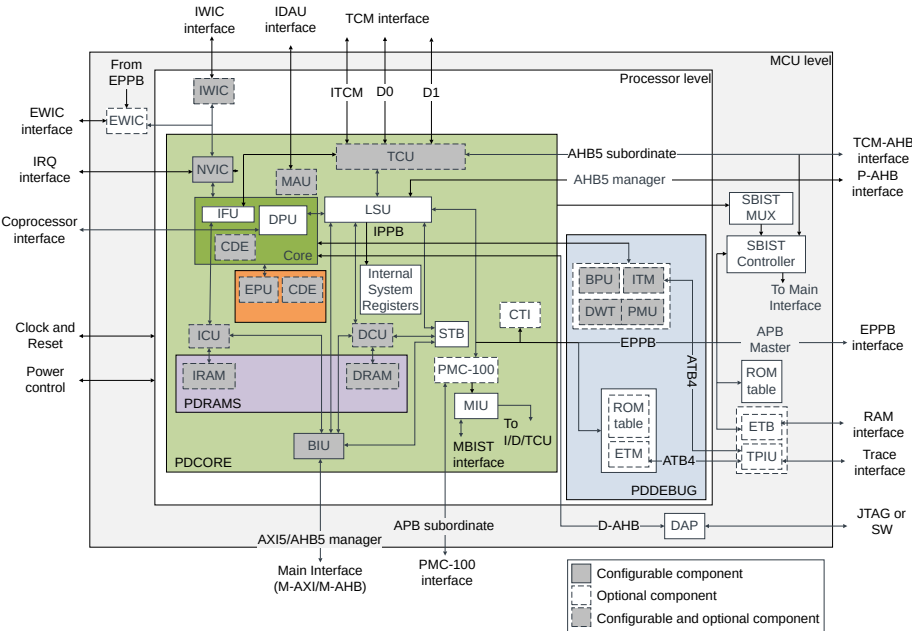
### 3. Technical overview

This chapter describes the Cortex®-M52 processor components and configuration options.

#### 3.1 Cortex®-M52 processor components

The Cortex®-M52 processor has fixed and optional component blocks.

**Figure 3-1: Cortex®-M52 processor has fixed and optional component blocks**



For more information about the PDCORE, PDDEBUG, and PDRAMS power domains, and their clocking, reset, and power requirements, see [Power management](#).

The following table describes the various processor components shown in the processor block diagram.

**Table 3-1: Processor components**

Block	Component
Processor core	The Cortex®-M52 processor core has an <i>Instruction Fetch Unit</i> (IFU) that is closely coupled with the <i>Data Processing Unit</i> (DPU). For more information, see <a href="#">Cortex-M52 processor core</a> .
Extension Processing Unit (EPU)	<p>The EPU performs:</p> <ul style="list-style-type: none"> <li>• Scalar floating-point operations</li> <li>• <i>M-class Vector Extension</i> (MVE) operations</li> </ul> <p>For more information, see <a href="#">Extension Processing Unit</a>. The EPU can be optionally included.</p>
Memory components	<p>The memory components are:</p> <ul style="list-style-type: none"> <li>• <i>Memory Authentication Unit</i> (MAU). For more information on the MAU, see <a href="#">Memory Authentication</a>. The MAU contains: <ul style="list-style-type: none"> <li>◦ <i>Security Attribution Unit</i> (SAU)</li> <li>◦ <i>TCM Gate Unit</i> (TGU)</li> <li>◦ Secure MPU region, MPU_S, which is always optionally configured</li> <li>◦ Non-secure MPU region, MPU_N, which is always optionally configured</li> </ul> </li> <li>• <i>Load Store Unit</i> (LSU)</li> <li>• <i>TCM Control Unit</i> (TCU)</li> <li>• <i>Data Cache Unit</i> (DCU) and <i>Data RAM</i> (DRAM).</li> <li>• <i>Instruction Cache Unit</i> (ICU) and <i>Instruction RAM</i> (IRAM)</li> <li>• <i>Bus Interface Unit</i> (BIU)</li> <li>• <i>STore Buffer</i> (STB)</li> <li>• <i>MBIST Interface Unit</i> (MIU)</li> </ul> <p>For more information on the memory system, see <a href="#">Memory system</a>.</p>
Interrupt components	<p>The interrupt components are:</p> <ul style="list-style-type: none"> <li>• <i>Nested Vectored Interrupt Controller</i> (NVIC)</li> <li>• <i>External Wakeup Interrupt Controller</i> (EWIC), which can be optionally included</li> <li>• <i>Internal Wakeup Interrupt Controller</i> (IWIC), which can be optionally included</li> </ul> <p>For more information on the interrupt-related components, see <a href="#">Interrupt components</a>.</p>

Block	Component
Debug and trace components	<p>The debug and trace components are:</p> <ul style="list-style-type: none"> <li>• <i>BreakPoint unit</i> (BPU)</li> <li>• <i>Cross Trigger Interface</i> (CTI), which is optionally configured</li> <li>• CoreSight™-compliant <i>Debug Access Port</i> (DAP), CoreSight™ DAP-Lite2, which is available for download when you license Cortex®-M52 processor IP.</li> <li>• <i>Data Watchpoint and Trace</i> (DWT) unit</li> <li>• <i>Performance Monitoring Unit</i> (PMU), which is located in the DWT</li> <li>• <i>Embedded Trace Macrocell</i> (ETM)</li> <li>• <i>Instrumentation Trace Macrocell</i> (ITM)</li> <li>• <i>Trace Port Interface Unit</i> (TPIU)</li> <li>• CoreSight™-compliant <i>Embedded Trace Buffer</i> (ETB) functionality support. The ETB is not delivered as a part of the IP deliverable. The ETB is an optional licensable component which is available when you license either the CoreSight™ SoC-600 or CoreSight™ SoC-600M. The Cortex®-M52 IP deliverable has a placeholder for ETB integration.</li> </ul> <p>For more information on the debug and trace related components, see <a href="#">Debug and trace components</a>.</p>



Note

- If the Cortex®-M52 processor is configured with minimal debug, then the ETM and ITM cannot be included.
- If the Cortex®-M52 processor is configured with reduced set or full set debug, then the ETM and ITM are optional.
- If the Cortex®-M52 processor is configured with the reduced set or the full set debug, then the BPU and DWT are always included.

**Table 3-2: Processor components related to functional safety and testing**

Block	Component
Testing components	<p>The testing components are:</p> <ul style="list-style-type: none"> <li>• <i>Programmable MBIST Controller</i> (PMC-00)</li> <li>• <i>Software Built-In Self-Test</i> (SBIST) components</li> </ul>

### 3.1.1 Cortex®-M52 processor core

The Cortex®-M52 processor core has an *Instruction Fetch Unit* (IFU) that is closely coupled with the *Data Processing Unit* (DPU).

The DPU contains the logic to:

- Decode and execute scalar integer instructions
- Handle the register transfer operations required for exception entry and exit

The Cortex®-M52 processor core has the following features:

- An in-order four-stage integer pipeline.

- Two *Arithmetic Logic Units* (ALUs):
  - One ALU for simple arithmetic operations.
  - One ALU that can handle shift and the SIMD operations included in the *Digital Signal Processing* (DSP) Extension.
- The core can handle only one 32-bit load operation, when *M-profile Vector Extension* (MVE) is configured in the Cortex®-M52 processor.
- Harvard bus interfaces with vector fetch capability on the instruction side to optimize exception entry for efficient operation of compute workloads.
  - 32-bit instruction fetch data width.
  - 32-bit load/store data width.
- Optimized set of integer register bank ports for energy-efficient operation.
- Integer divide unit with support for operand-dependent early termination. In this context, early termination refers to operations that terminate sooner than the expected number of cycles for the integer divide unit. Early termination capabilities depend on the data that enters the pipeline.
- Single cycle branch latency in most instances, without a requirement for branch prediction.
- Limited dual-issue of common 16-bit instruction pairs.
- Support for exception-continuable load and store multiple accesses.
- Instruction queue to decouple instruction fetching and instruction execution.



The Cortex®-M52 processor core works with the *Extension Processing Unit* (EPU), when configured to provide support for:

- Integer and floating-point operations included in MVE
- Depend on the configuration, support scalar half-precision, single-precision floating-point operations, or half-precision, single-precision, and double-precision floating-point operations.

To support *Arm Custom Instructions* (ACIs), the processor includes optional *Custom Datapath Extension* (CDE) modules, which are embedded inside the logic. These modules are used to execute user-defined instructions that work on general-purpose integer, floating point, and MVE registers.

### 3.1.2 Extension Processing Unit

The *Extension Processing Unit* (EPU) includes support for all the instructions in the *M-profile Vector Extension* (MVE) and half, single, and double-precision scalar FPU architecture.

The EPU has the following features:

- MVE is implemented using a 32-bit arithmetic and load/store data-path in a one beats per tick configuration. A beat is the execution of  $\frac{1}{4}$  of an MVE instruction. Instructions can overlap to allow full utilization of the logic with a sustained bandwidth of 32-bit *Multiply Accumulate* (MAC)



and 32-bit load/store per cycle. For more information on vector operation terminology, see *Arm®v8-M Architecture Reference Manual*.

- Extended register file, which is optimized for efficient vector operations.
- Floating-point MAC unit capable of a throughput of one single-precision or two-half precision MAC instructions every cycle when MVE is included in the Cortex®-M52 processor, or one single or half-precision MAC every cycle when only scalar floating-point is configured.
- Area optimized double-precision floating-point implementation.
- Support for Security Extension including lazy context stacking.

To support *Arm Custom Instructions* (ACIs), the EPU includes a floating-point and MVE CDE module. This module is used to execute user-defined instructions that work on floating-point and MVE registers. If the optional EPU is not present, then the optional floating-point CDE module is not present.

### 3.1.3 Memory components

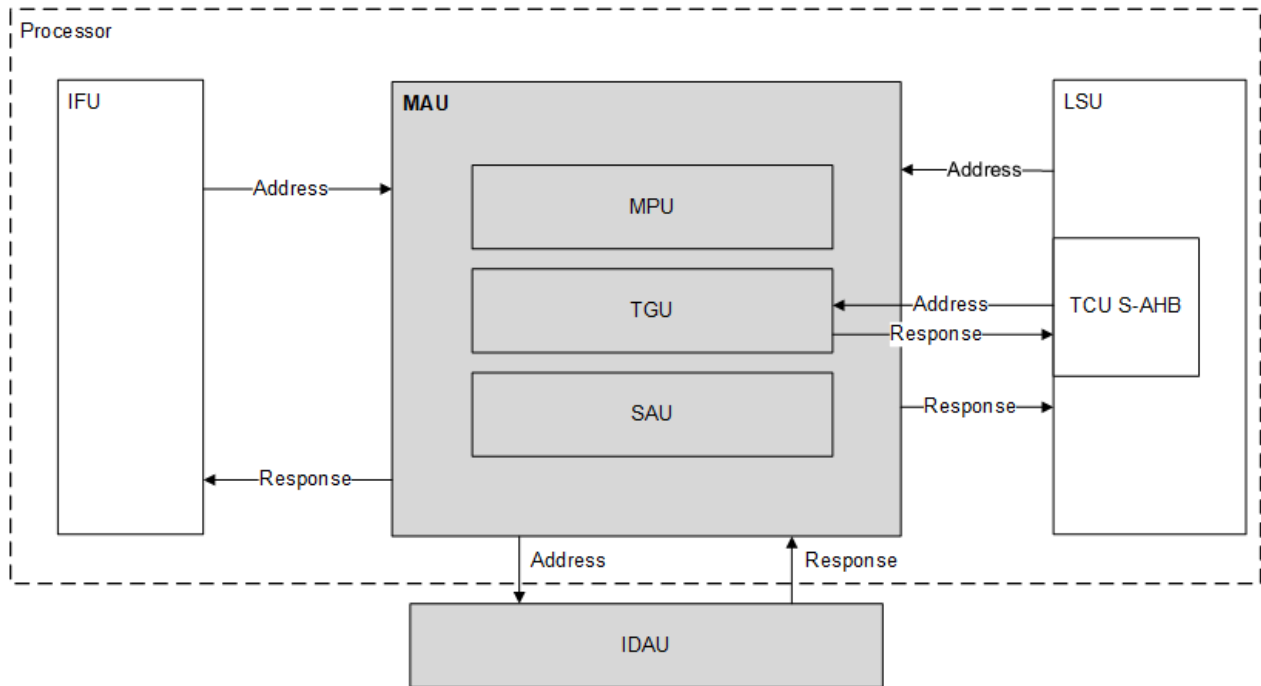
The Cortex®-M52 processor memory components consist of the *Memory Authentication Unit* (MAU) and memory system interfaces.

#### Memory Authentication Unit

The Cortex®-M52 processor *Memory Authentication Unit* (MAU) contains several units that control access to the memory.

The following figure shows the MAU block diagram.

**Figure 3-2: MAU block diagram**



### Memory Protection Unit

The *Memory Protection Unit* (MPU) supports the Arm® *Protected Memory System Architecture* (PMSA). Therefore, the MPU provides programmable support for memory protection using many software controllable regions. This unit defines the memory attributes that are associated with a particular memory region and the access permissions of addresses. Memory regions can be programmed to generate faults when accessed inappropriately, for example, by unprivileged software, reducing the scope of incorrectly written application code. The architecture includes fault status registers to allow an exception handler to determine the source of the fault and to apply corrective action or notify the system.

If the Security Extension is implemented, the entire MPU logic can be split into Secure and Non-secure MPU regions.

### Security Attribution Unit

The *Security Attribution Unit* (SAU) defines and authenticates accesses to memory based on the Security state of the core or the debugger. These states can be any of the following:

- Non-secure.
- Secure and Non-secure Callable.
- Secure.

### TCM Gate Unit

The *TCM Gate Unit* (TGU) controls software and *AHB TCM Access* (TCM-AHB) accesses to the TCMs based on the security attribute of the access.

## Interface to the IDAU

The MAU contains an interface to the *Implementation Defined Attribution Unit* (IDAU), which is present outside the core and not a part of the Cortex®-M52 processor. This unit defines memory regions as being either Secure, Non-secure, Non-secure Callable, or exempt from security checking. The final security mapping of memory regions is a combination of the response from the SAU and IDAU.

## Memory system

The Cortex®-M52 processor memory system provides the interface between the core and the caches, external memory interfaces, and internal memory-mapped registers.

The memory system includes:

- A single interface to an *Instruction Tightly Coupled Memory* (ITCM) and two interfaces to *Data Tightly Coupled Memories* (DTCMs), D0TCM, and D1TCM.
- A configurable Main interface that can be used for on-chip or off-chip memory and devices. Main interface can either be AXI Main (M-AXI) interface or AHB Main (M-AHB) interface depending on your configuration. M-AHB is comprised of code region AHB Main interface (CODE-AHB) and system region AHB Main interface (SYS-AHB).
- A *Peripheral AHB* (P-AHB) interface for access to external peripherals
- A *AHB TCM Access* (TCM-AHB) interface for system access to the TCMs
- An L1 instruction cache
- An L1 data cache
- An L1 unified cache (exclusive to L1 instruction and data cache)
- An *External Private Peripheral Bus* (EPPB) interface for CoreSight™ debug and trace components
- A *Store Buffer* (STB) to hold store operations targeting Main interface when they have left the load/store pipeline and the DPU has committed them. From the STB, a store can do any of the following items:
  - Request access to the cache RAM through the DCU
  - Request the *Bus Interface Unit* (BIU) to initiate linefills
  - Request the BIU to write data on the Main interface
  - Forward store data to load on pipeline
- A *Store Queue* (SQ) to hold store operations targeting TCM (ITCM and DTCMs) when they have left the load/store pipeline and the DPU has committed them. From the SQ, a store can request access to the instruction or data TCM.

If there are several store transactions that are associated with the same 32-bit aligned word, the STB can merge these store transactions into a single transaction.

For more information, see:

- [Memory model](#).
- [Memory system](#).

### 3.1.4 Interrupt components

The Cortex®-M52 processor interrupt components are responsible for low-latency interrupt processing and enabling the Cortex®-M52 processor to enter and wake up from low-power state.

#### NVIC features

The Cortex®-M52 processor *Nested Vectored Interrupt Controller* (NVIC) is closely integrated with the core to achieve low-latency interrupt processing.

The NVIC is responsible for:

- Maintaining the current execution priority of the Cortex®-M52 processor.
- Maintaining the pending and active status of all exceptions that are supported.
- Invoking preemption when a pending exception has priority.
- Providing wakeup signals to wakeup the Cortex®-M52 processor from deep sleep mode.
- Providing support to the *Internal Wakeup Interrupt Controller* (IWIC) and *External Wakeup Interrupt Controller* (EWIC).
- Providing priority and exception information to other processor components.

The NVIC in the Cortex®-M52 processor allows up to 496 exceptions, of which, 480 can be regular external interrupts.

#### Wakeup Interrupt Controller

The Cortex®-M52 processor supports a *Wakeup Interrupt Controller* (WIC) unit that allows the Cortex®-M52 processor to enter low-power state.

Two WICs that are supported:

- An *Internal Wakeup Interrupt Controller* (IWIC) that is synchronous with the processor and contained within the Cortex®-M52 processor boundary.
- An *External Wakeup Interrupt Controller* (EWIC), which is a system-level component that can be asynchronous to the Cortex®-M52 processor.

The Cortex®-M52 processor supports any of the following:

- No WIC.
- IWIC only.
- EWIC only.
- Both IWIC and EWIC.

### 3.1.5 Debug and trace components

The Cortex®-M52 processor has optional and configurable debug and trace components.

#### BreakPoint Unit

A configurable *BreakPoint Unit* (BPU) for implementing breakpoints.

#### Data Watchpoint and Trace

A configurable *Data Watchpoint and Trace* (DWT) unit for implementing watchpoints, data tracing, and system profiling.

#### Instrumentation Trace Macrocell

An optional *Instrumentation Trace Macrocell* (ITM) that supports `printf()` style debugging using instrumentation trace.

#### Performance Monitoring Unit

A *Performance Monitoring Unit* (PMU) which enables software and debugger to gather statistics on events taking place on the Cortex®-M52 processor. These statistics can be used for performance analysis and system debug.

The PMU is always present when the DWT is present.

#### ROM tables

ROM tables allow debuggers to determine which CoreSight™ components are implemented in the Cortex®-M52 processor.

#### Debug and trace interfaces

These interfaces are suitable for:

- Passing on-chip data through a *Trace Port Interface Unit* (TPIU) to a *Trace Port Analyzer* (TPA), including *Serial Wire Output* (SWO) mode.
- Integrating a *Debug Access Port* (DAP), which is a debug port that is used to control debug functionality.
- Integrating a CoreSight™ *Embedded Trace Buffer* (ETB), which is an optional licensable component for trace data to be written to an external SRAM.

#### Cross Trigger Interface

The *Cross Trigger Interface* (CTI) enables the debug logic and *Embedded Trace Macrocell* (ETM) to interact with each other and with other CoreSight™ components.

#### Embedded Trace Macrocell

The optional ETM provides instruction-only trace capabilities. For more information, see the *Arm China CoreSight™ ETM-M52 Technical Reference Manual*.

### 3.1.6 Testing components

The Cortex®-M52 processor testing components perform on-line *Memory Built-In Self Test* (MBIST) and *Software Built-In Self Test* (SBIST) to test functional logic.

#### PMC-100

PMC-100 is an optional on-line *Memory Built-In Self Test* (MBIST) controller that is used to test RAMs, *Error Correcting Code* (ECC) logic, and any other associated logic.

#### SBIST controller

The *Software Built-In Self Test* (SBIST) controller is an optional component that is used to facilitate the testing of functional logic (excluding memories).

## 3.2 Interfaces

The following table summarizes the interfaces that the Cortex®-M52 processor supports.



Note

For more information on the protocols in the following table, refer to the following specifications:

- Arm® AMBA® 5 AHB Protocol Specification
- AMBA® APB Protocol Version 2.0 Specification
- AMBA® 4 ATB Protocol Specification
- AMBA® AXI and ACE Protocol Specification

The following key is used in [Interfaces](#):

**Table 3-3: Interfaces**

Name	Protocol	Width	Details
AXI Main (M-AXI) interface	Compliant with AMBA® 5 AXI protocol	32-bit	Provides access to memory and peripheral components in the system. Use either M-AXI or M-AHB depending on your configuration.
AHB Main (M-AHB) interface	Compliant with AMBA® 5 AHB protocol	32-bit	Provides access to memory and peripheral components in the system. Use either M-AXI or M-AHB depending on your configuration.
<i>Instruction Tightly Coupled Memory</i> (ITCM) and <i>Data Tightly Coupled Memory</i> (DTCM)	-	ITCM: 32-bit  DTCM: 2 banks of 32-bits	One ITCM interface and two DTCM interfaces to provide high-bandwidth access from the Cortex®-M52 processor and <i>AHB TCM Access</i> (TCM-AHB) interface to local low-latency memory. The size of both TCM instances is configurable in the range of 4KB-16MB in powers of 2. The Cortex®-M52 processor also supports zero size TCMs.

Name	Protocol	Width	Details
AHB TCM Access (TCM-AHB) interface	AMBA 5 AHB	32-bit	Provides system access to the TCMs. A <i>Direct Memory Access</i> (DMA) engine typically uses this interface.
Tightly coupled <i>Peripheral AHB</i> (P-AHB) interface	AMBA 5 AHB	32-bit	Provides access to system peripherals.
<i>External Private Peripheral Bus</i> (EPPB) interface	AMBA 4 APB	32-bit	Used to connect to external CoreSight™-compliant peripherals.
PMC-100 external (PMC-100 APB) interface	AMBA 4 APB slave interface	32-bit	Provides direct external access to the <i>PMC-100 Programmable MBIST Controller</i> .
Lockstep interface	-	-	Used to control <i>Dual-Core Lock-step</i> (DCLS) and report on comparator match and other errors
External IDAU interface	-	-	Allows the system to define security attributes.
ITM and ETM interfaces	AMBA 4 ATB	8-bit	Provides tracing capability.
Coprocessor interface	-	64-bit	Used for closely-coupled external accelerator hardware.
<i>Debug AHB</i> (D-AHB) slave interface	AMBA 5 AHB	32-bit	Provides debug access to registers, memory, and peripherals.
<i>Cross Trigger Interface</i> (CTI) interface	-	Four channels	Used for debug and trace synchronization. The CTI is optional, however the CTI interface is always present.
Power control interface	P-Channel and Q-Channel	-	Optional support for several internal power domains which can be enabled and disabled using the P-Channel and Q-Channel interfaces connected to a power controller in the system. For more information, see <a href="#">Power management</a> or the <i>Arm China Cortex®-M52 Processor Integration and Implementation Manual</i> . The <i>Arm China Cortex®-M52 Processor Integration and Implementation Manual</i> is only available to licensees.
<i>External Wakeup Interrupt Controller</i> (EWIC) interface.	-	-	Provides access to an optional EWIC, which is a peripheral to the system and is suitable for sleep states where the entire processor sub-system is powered down.

### 3.3 Security

Arm® TrustZone® uses the Security Extension, which supports Secure and Non-secure states on all memory interfaces, including security gating on *Tightly Coupled Memory* (TCM) interfaces.

Memory and peripherals in the system can be marked as Secure, making them accessible only to code that is running in the Secure state.

Interrupts can be marked as Secure indicating that they are handled by Secure handler code in the Secure world.

Hardware protects all Secure resources, including firmware and sensitive data values from being visible to Non-secure code and debug. If you are programming in Secure state, you can choose which Secure functions can be called by Non-secure code, where the Secure functions can tightly control the parameters of such function calls.

## 3.4 Functional safety and reliability

The following are the Cortex®-M52 processor functional safety and reliability features.

- L1 cache and TCM interfaces support optional internal *Error Correcting Code* (ECC). All ECC errors are reported to the system on an external interface.
- *Reliability, Availability, and Serviceability* (RAS) Extension support.
- Optional interface protection included on the Main interface, TCM-AHB, P-AHB, *Debug AHB* (D-AHB), and EPPB interfaces.
- *Dual-Core Lock-Step* (DCLS) operation is supported. In DCLS configurations, there is a second, redundant copy of the majority of the processor core and *Internal Wakeup Interrupt Controller* (IWIC) logic. All inputs to the logic are duplicated and connected to both copies of the logic. The outputs from the two copies of logic are compared for errors. Faults can occur in either copy of the logic and cause errors on the outputs, however, comparators cannot determine whether the primary or redundant copy of logic is faulty.
- Optional *Programmable MBIST Controller* (PMC-100) for embedded memory and ECC logic testing during processor run-time. For more information, see the *Arm® PMC-100 Technical Reference Manual*. The processor also supports direct access to the PMC-100 from an external agent in the system through an AMBA® 4 APB slave interface as described in [PMC-100 interface signals](#). Access to the PMC-100 on this interface is only permitted for requests marked as secure and privileged in PMCPPROT.
- Optional licensable *Software Test Library* (STL), which is designed to provide maximum fault coverage in a compact ROM image with short runtime. The processor contains observation points in the *Nested Vectored Interrupt Controller* (NVIC) and *Memory Protection Unit* (MPU) which can be used by the STL to improve fault coverage and reduce the number of instructions required in the tests. The library also uses PMC-100 to test the ECC and memory system of the processor. The MCU layer includes support for an optional SBIST controller unit and associated SBIST components which are used by the library code to control and monitor the test. The SBIST controller and associated SBIST components are delivered with the processor. For more information on the SBIST controller and associated SBIST components, see the *Arm China Cortex®-M52 Processor Integration and Implementation Manual*. The *Arm China Cortex®-M52 Processor Integration and Implementation Manual* is a confidential document that is available only to licensees and partners with an NDA agreement.



## 3.5 Power intent

The Cortex®-M52 processor power intent features include:

- Support for multiple power domain *State Retention Power Gating* (SRPG) implementation through *Unified Power Format* (UPF). The UPF files are IEEE 1801-2009 compliant.
- Power control based on the Arm® standard P-Channel and Q-Channel interfaces. For information on the P-Channel and Q-Channel logic interfaces, see *AMBA® Low Power Interface Specification Arm® Q-Channel and P-Channel Interfaces*.
- Support for an *Internal Wakeup Interrupt Controller* (IWIC) and an *External Wakeup Interrupt Controller* (EWIC).

## 3.6 Performance considerations

Software can help to optimize the performance of the Cortex®-M52 processor.

To get the best performance out of the Cortex®-M52 processor, software can enable loop and branch info cache. By default, the *Low Overhead Branch* (LOB) feature is disabled after reset. To enable this feature, software can set the LOB bit in the *Configuration and Control Register* (CCR), and then execute an ISB instruction. For more information on the CCR, see [System control register summary](#)

## 3.7 Cortex®-M52 implementation options

The Cortex®-M52 processor has configurable options that the chip designer can set during the implementation and integration stages to match your functional requirements.

The following table shows the Cortex®-M52 processor configurable options available at implementation time.

**Table 3-4: Cortex®-M52 processor configurable options**

Feature	Options
Floating-point and <i>M-profile Vector Extension</i> (FPMVE) support	<p>The floating-point and MVE features together specify the MVE functionality that is supported on the Cortex®-M52 processor. Floating-point and MVE functionality can either be included or excluded.</p> <ul style="list-style-type: none"> <li>• MVE not included. Scalar HP/SP included.</li> <li>• MVE not included. Scalar HP/SP/DP included.</li> <li>• FP not included. Integer subset of MVE (MVE-I) included.</li> <li>• Integer subset of MVE (MVE-I) and Scalar HP/SP included.</li> <li>• Integer MVE (MVE-I), HP and SP MVE (MVE-F), and Scalar HP/SP/DP are included.</li> </ul>
Inclusion of Security Extension	No Security Extension present
	Security Extension present

Feature	Options
Inclusion of AXI or AHB	M-AXI present
	CODE-AHB and SYS-AHB present
Inclusion of PACBTI Extension	No PACBTI Extension present
	PACBTI Extension present
Coprocessor support	No support for coprocessor hardware
	Support for coprocessor hardware
Inclusion of Non-secure <i>Memory Protection Unit</i> (MPU)	0 region, 4 regions, 8 regions, 12 regions, or 16 regions
Inclusion of Secure <i>Memory Protection Unit</i> (MPU)	0 region, 4 regions, 8 regions, 12 regions, or 16 regions when the Security Extension is included.
Inclusion of <i>Security Attribution Unit</i> (SAU)	0 region, 4 regions, or 8 regions when the Security Extension is included.
Inclusion and size of Unified cache	No Unified cache. When Ucache is not configured, Icache and Dcache can be configured independently.
	Unified cache is configured. Unified cache included and the size can be 1KB, 2KB, 4KB, 8KB, 16KB, 32KB, or 64KB. When Ucache is configured, no Icache or Dcache exists.
Inclusion and size of instruction cache	No <i>Instruction Cache Unit</i> (ICU)
	ICU included and the size can be 4KB, 8KB, 16KB, 32KB, or 64KB.
Inclusion and size of data cache	Area optimized Main interface, no <i>Data Cache Unit</i> (DCU).
	DCU included and the size can be 4KB, 8KB, 16KB, 32KB, or 64KB.
Inclusion of <i>Error Correcting Code</i> (ECC)	No ECC on caches or TCMs
	ECC on all implemented caches and TCMs
Number of interrupts	1-480 interrupts. To support non-contiguous mapping, you can remove individual interrupts.
Instruction and data cache ID	Unique identifier for instruction and data cache RAM implementation, and can take values range from 0-255.
Number of exception priority bits	3-8 priority bits.
Disable support for individual interrupts	When set to 1, support for individual interrupts is disabled, therefore, allowing a range of non-contiguous interrupts.
Debug resources included. This feature also controls the number of <i>Performance Monitoring Unit</i> (PMU) counters that are present.	Minimal debug. No Halting debug or memory and peripheral access.
	Reduced set. Two data watchpoint comparators and four breakpoint comparators.
	Mid set. Four data watchpoint comparators and eight breakpoint comparators.
	Full set. Eight data watchpoint comparators and eight breakpoint comparators.
Inclusion of <i>Instrumentation Trace Macrocell</i> (ITM) and <i>Data Watchpoint and Trace</i> (DWT) trace	No ITM or DWT trace
	Complete ITM and DWT trace
Inclusion of <i>Embedded Trace Macrocell</i> (ETM)	No ETM support
	ETM instruction execution trace
Inclusion of <i>Cross Trigger Interface</i> (CTI)	No CTI
	CTI is included
Inclusion of <i>Internal Wakeup Interrupt Controller</i> (IWIC)	No IWIC
	IWIC is included

Feature	Options
Number of IRQ lines supported by the IWIC and EWIC	The value always includes the three internal events NMI, RXEV, Debug monitor event, and at least one IRQ.
Inclusion of interface protection	No interface protection
	Interface protection is included. Interface protection provides parity bits to the bus interface to help with fault coverage in functional safety applications.
Inclusion of lockstep operation	No lockstep operation
	<i>Dual Core Lockstep</i> (DCLS) operation included
Inclusion of ITCM security gating	No ITCM security gate
	ITCM security gate included
ITCM security gate block size in bytes	$2^{(\text{Instruction TCM Gate Unit (TGU) block size}+5)}$
Number of ITCM security gate blocks	$2^{\text{Maximum number of instruction TGU blocks}}$
Inclusion of DTCM security gating	No DTCM security gate
	DTCM security gate included
DTCM security gate block size in bytes	$2^{(\text{Data TGU block size}+5)}$
Number of DTCM security gate blocks	$2^{\text{Maximum number of data TGU blocks}}$
PMC-100 support	No <i>Programmable MBIST Controller</i> (PMC-100)
	PMC-100 included
Number of PMC-100 program registers	Specifies the number of program registers implemented in PMC-100. Values 0 and 1 are reserved. The range is 2-32.
Reset all registers functionality	Specifies whether all synchronous states or only the architecturally required states are reset. <ul style="list-style-type: none"> <li>Only reset states that architecture requires.</li> <li>Reset all synchronous states.</li> </ul>
Arm Custom Instructions (ACIs) with Custom Datapath Extension (CDE) modules on a coprocessor basis	If CDE is not included for CP<n> instructions in the CP<n> encoding space, these instructions are executed on the coprocessor interface and the CDE modules are not used. If CDE is implemented for CP<n> instructions in the CP<n> encoding space, these instructions are executed by a CDE module and the coprocessor interface is not used.
Flop parity protection	No flop parity protection
	Flop parity protection included
SBIST relevant observation register	No SBIST observation register
	SBIST observation registers included



Note

- The parameter to control inclusion of the *External Wakeup Interrupt Controller* (EWIC) can be configured at the MCU level. The MCU level supports all the processor-level configuration and contains additional configuration parameters to configure the functionality that is specific to CoreSight™ components that are included in the system.
- Signal tie-offs determine the inclusion of the ITCM and DTCM.
- Additionally, there are static and reset configuration signals. For more information, see [Static configuration signals](#) and [Reset configuration signals](#).

## 4. Programmers model

This chapter describes the Cortex®-M52 processor register set, modes of operation, and provides information on programming the Cortex®-M52 processor.

The Cortex®-M52 programmers model is an implementation of the Main Extension architecture. For a complete description of the programmers model, see the *Arm®v8-M Architecture Reference Manual*.

### 4.1 Security states, operation, and execution modes

The Cortex®-M52 processor supports Secure and Non-secure Security states, Thread and Handler operating modes, and can run in either Thumb or Debug operating states. In addition, the Cortex®-M52 processor can limit or exclude access to some resources by executing code in privileged or unprivileged mode.

See the *Arm®v8-M Architecture Reference Manual* for more information about the modes of operation and execution.

#### Security states

When the Security Extension is included in the Cortex®-M52 processor, the programmers model includes two orthogonal Security states, Secure state and Non-secure state. This means the processor is in Secure or Non-secure state, but not both at the same time. When the Security Extension is implemented, the Cortex®-M52 processor always resets into Secure state. When the Security Extension is not implemented, the Cortex®-M52 processor resets into Non-secure state. Each Security state includes a set of independent operating modes and supports both privileged and unprivileged user access. Registers in the *System Control Space* (SCS) are banked across Secure and Non-secure state, with the Non-secure register view available at an aliased address to Secure state.

When the Security Extension is not included in the Cortex®-M52 processor, the programmers model includes only the Non-secure state.

#### Operating modes

For each Security state, the Cortex®-M52 processor can operate in Thread or Handler mode. The conditions which cause the Cortex®-M52 processor to enter Thread or Handler mode are as follows:

- The Cortex®-M52 processor enters Thread mode on reset, or as a result of an exception return to Thread mode. The Thread mode supports both privileged and unprivileged execution.
- The Cortex®-M52 processor enters Handler mode as a result of an exception. The Handler mode only supports privileged execution.

The Cortex®-M52 processor can change Security state on taking an exception. For example, when a Secure exception is taken from Non-secure state Thread or Handler mode, the Cortex®-M52 processor enters the Secure state Handler mode.

The Cortex®-M52 processor can also call Secure functions from Non-secure state and Non-secure functions from Secure state. The Security Extension includes requirements for these calls to prevent secure data from being accessed in Non-secure state.

### Operating states

The Cortex®-M52 processor can operate in T32 or Debug state:

- T32 state is the state of normal execution running 16-bit and 32-bit halfword-aligned T32 instructions.
- Debug state is the state when the Cortex®-M52 processor is in Halting debug.

### Privileged access and unprivileged user access

Code can execute as privileged or unprivileged. Unprivileged execution limits or excludes access to some resources appropriate to the current Security state. Privileged execution has access to all resources available to the Security state. Handler mode is always privileged. Thread mode can be privileged or unprivileged.

## 4.2 Instruction set summary

The Cortex®-M52 processor implements the Arm® instruction set.

These instructions include:

- All base instructions
- All instructions in the Main Extension
- All instructions in the *Digital Signal Processing* (DSP) Extension
- All instructions in the PACBTI Extension
- Optionally some of the coprocessor instructions:
  - CDP, CDP2
  - MCR, MCR2
  - MCRR, MCRR2
  - MRC, MRC2
  - MRRC, MRRC2
- Optionally all instructions in the Security Extension
- Optionally all half-precision, single-precision, and double-precision instructions in the Floating-point Extension
- Optionally all vector operation instructions on integer operations in the *M-profile Vector Extension* (MVE)
- Optionally all vector operation instructions on half-precision and single-precision floating-point operations in MVE
- Optionally all the *Reliability, Availability, and Serviceability* (RAS) Extension instructions

For more information about these instructions, see the *Arm®v8-M Architecture Reference Manual*.

The processor also implements *Custom Datapath Extension* (CDE) instructions. The CDE introduces three classes of instructions in the coprocessor instruction space:

- Three instructions operate on the general-purpose register file.
- Three instructions operate on the floating-point register file.
- Three instructions operate on the MVE register file.

For specific information on the CDE instructions implemented in the processor, see [Arm Custom Instructions](#). For general information on CDE instructions, see the *Arm®v8-M Architecture Reference Manual*.

## 4.3 Exclusive monitor

The Cortex®-M52 processor implements a local exclusive monitor contained in the *Load Store Unit* (LSU). The local monitor within the Cortex®-M52 processor has been constructed not to hold any physical address, but instead treats any store-exclusive access as matching the address of the previous load-exclusive.

This means that the implemented exclusives reservation granule is the entire memory address range. The TCMs support a local exclusive monitor, but not shared or global exclusive monitors. This implies that the TCMs support for exclusive requests between threads running on the Cortex®-M52 processor, but not exclusive requests between the Cortex®-M52 processor and a DMA (through the S-AHB). If an exclusive read access is carried out to a region which does not support a global monitor it must respond accordingly with either HEXOKAY LOW or RRESP[1:0] OKAY. These responses result in the transaction completing without setting the internal exclusive monitor. A subsequent exclusive store instruction does not carry out any memory transactions and sets the destination register to 1 indicating the exclusive access failed.

The external bus interfaces support an external exclusive monitor in the system to be shared with other bus masters.

For more information about semaphores and the local exclusive monitor, see the *Arm®v8-M Architecture Reference Manual*.

## 4.4 Cortex®-M52 processor core registers summary

The Cortex®-M52 processor core registers are 32 bits wide.

When the Security Extension is included, some of the registers are banked. The Secure view of these registers is available when the processor is in Secure state. The Non-secure view is available when the processor is in Non-secure and Secure state.

The following table shows the processor core register set summary. See the *Arm®v8-M Architecture Reference Manual* for information about the Cortex®-M52 processor core registers and their addresses, access types, and reset values.

**Table 4-1: Processor core register set summary**

Name	Description
R0-R12	R0-R12 are general-purpose registers for data operations.
MSP (R13)	<p>The stack pointer, SP, is register R13. In Thread mode, the CONTROL register indicates the stack pointer to use, main stack pointer, MSP, or process stack pointer, PSP.</p> <p>When the Security Extension is included, there are two MSP registers in the Cortex®-M52 processor:</p> <ul style="list-style-type: none"> <li>MSP_NS for the Non-secure state</li> <li>MSP_S for the Secure state</li> </ul> <p>When the Security Extension is included, two PSP registers exist in the Cortex®-M52 processor:</p> <ul style="list-style-type: none"> <li>PSP_NS for the Non-secure state</li> <li>PSP_S for the Secure state</li> </ul>
PSP (R13)	
MSPLIM	<p>The stack limit registers limit the extent to which the MSP and PSP registers can descend respectively.</p> <p>When the Security Extension is included, there are two MSPLIM registers in the Cortex®-M52 processor:</p> <ul style="list-style-type: none"> <li>MSPLIM_NS for the Non-secure state</li> <li>MSPLIM_S for the Secure state</li> </ul> <p>When the Security Extension is included, there are two PSPLIM registers in the Cortex®-M52 processor:</p> <ul style="list-style-type: none"> <li>PSPLIM_NS for the Non-secure state</li> <li>PSPLIM_S for the Secure state</li> </ul>
PSPLIM	
LR (R14)	The Link Register, LR, is register R14. It stores the return information for subroutines, function calls, and exceptions.
PC (R15)	The Program Counter, PC, is register R15. It contains the current program address.
XPSR	<p>The Program Status Register, PSR, combines:</p> <ul style="list-style-type: none"> <li>Application Program Status Register, APSR</li> <li>Interrupt Program Status Register, IPSR</li> <li>Execution Program Status Register, EPSR</li> </ul> <p>These registers provide different views of the PSR.</p>
PRIMASK	<p>The PRIMASK register prevents activation of exceptions with configurable priority. For information about the Exception model the Cortex®-M52 processor supports, see <a href="#">Exceptions</a>.</p> <p>There are two PRIMASK registers in the Cortex®-M52 processor:</p> <ul style="list-style-type: none"> <li>PRIMASK_NS for the Non-secure state</li> <li>PRIMASK_S for the Secure state</li> </ul>
BASEPRI	<p>The BASEPRI register defines the minimum priority for exception processing.</p> <p>There are two BASEPRI registers in the Cortex®-M52 processor:</p> <ul style="list-style-type: none"> <li>BASEPRI_NS for the Non-secure state</li> <li>BASEPRI_S for the Secure state</li> </ul>

Name	Description
FAULTMASK	<p>The FAULTMASK register prevents activation of all exceptions except for non-maskable interrupt, NMI and optionally Secure HardFault.</p> <p>Two FAULTMASK registers exist in the Cortex®-M52 processor:</p> <ul style="list-style-type: none"> <li>FAULTMASK_NS for the Non-secure state</li> <li>FAULTMASK_S for the Secure state</li> </ul>
LO_BRANCH_INFO	Loop and branch tracking information. Software cannot access LO_BRANCH_INFO.
SP	Current stack pointer register. SP_NS for the Non-secure state.
FPSCR	Floating-point Status and Control Register
S0-S31 / D0-15 / Q0-Q7	<p>S0-S31 are 32 single-precision floating-point registers. These registers can also be treated as:</p> <ul style="list-style-type: none"> <li>16 double-precision floating-point registers (D0-D15)</li> <li>8 vector registers (Q0-Q7)</li> </ul> <p>The <i>Extension Processing Unit</i> (EPU) can be configured to perform floating-point and <i>M-profile Vector Extension</i> (MVE) operations. See <a href="#">Floating-point and MVE support</a>.</p>
VPR	Vector Predication Status and Control Register
CONTROL	<p>The CONTROL register controls the stack that is used, and optionally the privilege level, when the Cortex®-M52 processor is in Thread mode.</p> <p>Two CONTROL registers exist in the Cortex®-M52 processor:</p> <ul style="list-style-type: none"> <li>CONTROL_NS for the Non-secure state</li> <li>CONTROL_S for the Secure state.</li> </ul>
PAC_KEY	Eight pointer authentication key registers

## 4.5 Architectural registers

Architectural registers can be fully architectural or architectural with some **IMPLEMENTATION DEFINED** bit fields.

Information on fully architectural registers that are listed in this section, see the *Arm®v8-M Architecture Reference Manual*. If registers are architectural with **IMPLEMENTATION DEFINED** bit fields, the register summary table in this section links registers to their associated register description.

For more information on architectural registers, see:

- [System control register summary](#).
- [Identification register summary](#).
- [Cache identification register summary](#)



## 4.6 Exceptions

Exceptions are handled and prioritized by the Cortex®-M52 processor and the *Nested Vectored Interrupt Controller* (NVIC). In addition to architecturally defined behavior, the Cortex®-M52 processor implements advanced exception and interrupt handling that reduces interrupt latency and includes **IMPLEMENTATION DEFINED** behavior.

### Exception handling and prioritization

The Cortex®-M52 processor core and the *Nested Vectored Interrupt Controller* (NVIC) together prioritize and handle all exceptions.

When handling exceptions:

- All exceptions are handled in Handler mode.
- Processor state is automatically stored to the stack on an exception, and automatically restored from the stack at the end of the *Interrupt Service Routine* (ISR).
- The vector is fetched in parallel to the state saving, enabling efficient interrupt entry.

The Cortex®-M52 processor supports tail-chaining that enables back-to-back interrupts without the overhead of state saving and restoration.

SoC designers configure the number of interrupts and bits of interrupt priority, during implementation. Software can choose only to enable a subset of the configured number of interrupts, and can choose how many bits of the configured priorities to use.

When the Security Extension is included, exceptions can be programmed as either Secure or Non-secure. When an exception is taken, the Cortex®-M52 processor switches to the associated Security state. The priority of Secure and Non-secure exceptions can be programmed independently. It is possible to deprioritize Non-secure configurable exceptions using AIRCR.PRIS to enable Secure interrupts to take priority. When taking and returning from an exception, the register state is always stored using the stack pointer associated with the background Security state. When taking a Non-secure exception from Secure state, all the register states are stacked, and then the registers are cleared to prevent Secure data being available to the Non-secure handler. The vector table base address is banked between Secure and Non-secure state. VTOR\_S, contains the Secure vector table base address and VTOR\_NS contains the Non-secure vector table base address. These registers can be programmed by software and also initialized at reset by the system.

If the Security Extension is not included all exceptions are Non-secure and only VTOR\_NS is used to determine the vector table base address.

Vector table entries are compatible with interworking between Arm® and Thumb® instructions. This causes bit[0] of the vector value to load into EPSR.T, on exception entry. All populated vectors in the vector table entries must have bit[0] set. Creating a vector table entry with bit[0] clear generates an INVSTATE (Invalid state flag) fault on the first instruction of the handler corresponding to this vector.

Input signals INITSVTOR[31:7] and INITNSVTOR[31:7] define the Secure and Non-secure vector table base address, respectively. However, when the Security Extension is not implemented, INITNSVTOR[31:7] defines the vector table base address.



- The Cortex®-M52 processor abandons all multicycle instructions to take pending interrupts. For more information, see the following *Multicycle instructions* section.
- Load Multiple and Store Multiple operations are interruptible.

---

## Multicycle instructions

A multicycle instruction can take one or more clock cycles to complete.

Load Multiple and Store Multiple operations are examples of a multicycle instruction.



A single load that is issued but waiting (wait stated) on the bus is not considered a multicycle instruction. However, it is abandoned to take pending interrupts if it has higher priority.

---

## 5. System registers

This chapter describes the system registers for the Cortex®-M52 processor.

### 5.1 System control register summary

The system control registers are a combination of fully architectural and **IMPLEMENTATION DEFINED** 32-bit registers and can be set to control various processor features.

The following table shows a summary of the system control registers.

For more information on the architectural registers that are listed in the following table, see the *Arm®v8-M Architecture Reference Manual*.

**Table 5-1: System control register summary**

Address	Name	Type	Reset value	Description
0xE00ECFC	REVIDR	RO	0x00000000 <b>Note:</b> The value of REVIDR[3:0] is determined by the input signal REVIDRNUM as specified in <a href="#">Miscellaneous signals</a> .	<a href="#">REVIDR, Revision ID Register</a>
0xE00ED00	CPUID	RO	0x630FD242	<a href="#">CPUID, CPUID Base Register</a>
0xE00ED04	ICSR	RW	0x00000000	Interrupt Control and State Register
0xE00ED08	VTOR	RW	0xFFFFFFFF0 <b>Note:</b> Bits [31:7] of VTOR_S are based on INITSVTOR[31:7]. Bits [31:7] of VTOR_NS are based on INITNSVTOR[31:7].  The Secure version of this register does not exist if the Security Extension is not configured and only INITNSVTOR[31:7] exists.  Bits [6:0] are <b>RES0</b> .	Vector Table Offset Register
0xE00ED0C	AIRCR	RW	0xFA05X000 <b>Note:</b> Bit [15] of this register depends on input signal CFGBIGEND. Bits [14:0] reset to zero.	Application Interrupt and Reset Control Register
0xE00ED10	SCR	RW	0x00000000	System Control Register

Address	Name	Type	Reset value	Description
0xE000ED14	CCR	RW	0x00000201	Configuration and Control Register
0xE000ED18	SHPR1	RW	0x00000000	System Handler Priority Register 1
0xE000ED1C	SHPR2	RW	0x00000000	System Handler Priority Register 2
0xE000ED20	SHPR3	RW	0x00000000	System Handler Priority Register 3
0xE000ED24	SHCSR	RW	0x00000000	System Handler Control and State Register
0xE000ED28	CFSR	RW	0x00000000	Configurable Fault Status Register A 32-bit register comprising MMFSR, BFSR, and UFSR.
	MMFSR	RW	0x00	MemManage Fault Status Register
0xE000ED29	BFSR	RW	0x00	BusFault Status Register
0xE000ED2A	UFSR	RW	0x0000	UsageFault Status Register
0xE000ED2C	HFSR	RW	0x00000000	HardFault Status Register
0xE000ED30	DFSR	RW	0x00000000 Cold reset only.	Debug Fault Status Register
0xE000ED34	MMFAR	RW	UNKNOWN	MemManage Fault Address Register
0xE000ED38	BFAR	RW	UNKNOWN	BusFault Address Register

Address	Name	Type	Reset value	Description
0xE000ED3C	AFSR	RW	0x00000000	AFSR, Auxiliary Fault Status Register
0xE000ED40	ID_PFR0	RO	0x20000030 <b>Note:</b> ID_PFR0[31:28] indicates support for the RAS Extension. ID_PFR0[31:28] is 0b0010 indicating that version 1 is implemented.	ID_PFR0, Processor Feature Register 0
0xE000ED44	ID_PFR1	RO	0x000002X0 <b>Note:</b> ID_PFR1[7:4] indicates support for the Security Extension. If the Security Extension is supported, then ID_PFR1[7:4] is 0b0011. If the Security Extension is not included, then ID_PFR1[7:4] is 0b0000.	Processor Feature Register 1
0xE000ED48	ID_DFR0	RO	0x10X00000 <b>Note:</b> ID_DFR0[23:20] indicates support for debug architecture. If halting debug is implemented and either a reduced set or a full set of debug resources is configured, then ID_DFR0[23:20] is 0b0010. If halting debug is not supported and minimal debug is supported, then ID_DFR0[23:20] is 0b0000.	Debug Feature Register 0
0xE000ED4C	ID_AFR0	RO	0x0000XXXX  Depends on the CDEMAPPEDONCP and CDERTLID parameters. For more information on these parameters, see the <i>Arm China Cortex®-M52 Processor Integration and Implementation Manual</i> . The <i>Arm China Cortex®-M52 Processor Integration and Implementation Manual</i> is a confidential document that is only available to licensees and partners with an NDA agreement.	Auxiliary Feature Register 0
0xE000ED50	ID_MMFR0	RO	0x00111040 <b>Note:</b> ID_MFR0[23:20] indicates support of Auxiliary Control registers. ID_MFR0[19:16] indicates support of TCMs. ID_MFR0[15:12] indicates that two levels of Shareability are implemented. ID_MFR0[11:8] indicates that the Outermost Shareability is implemented as Non-cacheable. ID_MFR0[7:4] indicates PMSAv8 support. All other bits are <b>RES0</b> .	Memory Model Feature Register 0
0xE000ED54	ID_MMFR1	RO	0x00000000	Memory Model Feature Register 1
0xE000ED58	ID_MMFR2	RO	0x01000000 <b>Note:</b> ID_MFR2[27:24] indicates that WFI can stall. All other bits are <b>RES0</b> .	Memory Model Feature Register 2
0xE000ED5C	ID_MMFR3	RO	0x00000011 <b>Note:</b> ID_MFR3[11:8] indicates that branch prediction is not supported. ID_MFR3[7:4] indicates that set/way maintenance operations are supported. ID_MFR3[3:0] indicates that address and instruction cache invalidate maintenance operations are supported. All other bits are <b>RES0</b> .	Memory Model Feature Register 3

Address	Name	Type	Reset value	Description
0xE000ED60	ID_ISAR0	RO	0x011X3110 ID_ISAR0[19:16] depend on whether the external coprocessor interface is included in the processor. <ul style="list-style-type: none"><li>If the external coprocessor is not included, there is no coprocessor instruction support, except the FPU. The value of X is 0x0.</li><li>If the external coprocessor is included, coprocessor instruction support is included. The value of X is 0x4.</li></ul>	Instruction Set Attribute Register 0
0xE000ED64	ID_ISAR1	RO	0x02112000	Instruction Set Attribute Register 1
0xE000ED68	ID_ISAR2	RO	0x20232232	Instruction Set Attribute Register 2
0xE000ED6C	ID_ISAR3	RO	0x01111131	Instruction Set Attribute Register 3
0xE000ED70	ID_ISAR4	RO	0x01310132	Instruction Set Attribute Register 4
0xE000ED74	ID_ISAR5	RO	0x00000000	Instruction Set Attribute Register 5
0xE000ED78	CLIDR	RO	0xFFFF000X <b>Note:</b> CLIDR[31:21] and CLIDR[2:0] depend on the cache configuration of the processor.	CLIDR, Cache Level ID Register
0xE000ED7C	CTR	RO	<ul style="list-style-type: none"><li>If an instruction cache or data cache is included, then the reset value is 0x8303C003.</li><li>If an instruction cache or data cache is not included, then the reset value is 0x00000000.</li></ul>	Cache Type Register
0xE000ED80	CCSIDR	RO	0xFFFFFFFF <b>Note:</b> CCSIDR depends on the CSSELR setting and L1 cache configuration.	CCSIDR, Current Cache Size ID Register
0xE000ED84	CSSELR	RW	0x00000000	CSSELR, Cache Size Selection Register
0xE000ED88	CPACR	RW	0x00000000	Coprocessor Access Control Register
0xE000ED8C	NSACR	RW	0x00000000	Non-secure Access Control Register

## 5.2 Identification register summary

The Cortex®-M52 processor identification registers allow software to determine the features and functionality that are available. Each of these registers is 32 bits wide.

The following table shows a summary of the identification registers. For more information on the architectural registers that are listed in the following table, see the *Arm®v8-M Architecture Reference Manual*.

**Table 5-2: Identification register summary**

Address	Name	Type	Reset value	Description
0xE000ED00	CPUID	RO	0x630FD242	CPUID, CPUID Base Register
0xE000ED40	ID_PFR0	RO	0x20000030 <b>Note:</b> ID_PFR0[31:28] indicates support for the RAS Extension. ID_PFR0[31:28] is 0b0010 indicating that version 1 is implemented.	Processor Feature Register 0
0xE000ED44	ID_PFR1	RO	0x000002X0 <b>Note:</b> ID_PFR1[7:4] indicates support for the Security Extension. If the Security Extension is supported, then ID_PFR1[7:4] is 0b0011. If the Security Extension is not included, then ID_PFR1[7:4] is 0b0000.	Processor Feature Register 1
0xE000ED48	ID_DFR0	RO	0x10X00000 <b>Note:</b> ID_DFR0[23:20] indicates support for debug architecture. If halting debug is implemented and either a reduced set or a full set of debug resources is configured, then ID_DFR0[23:20] is 0b0010. If halting debug is not supported and minimal debug is supported, then ID_DFR0[23:20] is 0b0000.	Debug Feature Register 0
0xE000ED4C	ID_AFR0	RO	0x0000XXXX  Depends on the CDEMAPPEDONCP and CDERTLID parameters. For more information on these parameters, see the <i>Arm China Cortex®-M52 Processor Integration and Implementation Manual</i> . The <i>Arm China Cortex®-M52 Processor Integration and Implementation Manual</i> is a confidential document that is only available to licensees and partners with an NDA agreement.	Auxiliary Feature Register 0
0xE000ED50	ID_MMFR0	RO	0x00111040 <b>Note:</b> ID_MFR0[23:20] indicates support of Auxiliary Control registers. ID_MFR0[19:16] indicates support of TCMs. ID_MFR0[15:12] indicates that two levels of Shareability are implemented. ID_MFR0[11:8] indicates that the Outermost Shareability is implemented as Non-cacheable. ID_MFR0[7:4] indicates PMSAv8 support. All other bits are <b>RES0</b> .	Memory Model Feature Register 0

Address	Name	Type	Reset value	Description
0xE000ED54	ID_MMFR1	RO	0x00000000	Memory Model Feature Register 1
0xE000ED58	ID_MMFR2	RO	0x01000000 <b>Note:</b> ID_MMFR2[27:24] indicates that WFI can stall. All other bits are <b>RES0</b> .	Memory Model Feature Register 2
0xE000ED5C	ID_MMFR3	RO	0x00000011 <b>Note:</b> ID_MMFR3[11:8] indicates that branch prediction is not supported. ID_MMFR3[7:4] indicates that set/way maintenance operations are supported. ID_MMFR3[3:0] indicates that address and instruction cache invalidate maintenance operations are supported. All other bits are <b>RES0</b> .	Memory Model Feature Register 3
0xE000ED60	ID_ISAR0	RO	0x011X3110 ID_ISAR0[19:16] depend on whether the external coprocessor interface is included in the processor. <ul style="list-style-type: none"><li>If the external coprocessor is not included, there is no coprocessor instruction support, except the FPU. The value of X is 0x0.</li><li>If the external coprocessor is included, coprocessor instruction support is included. The value of X is 0x4.</li></ul>	Instruction Set Attributes Register 0
0xE000ED64	ID_ISAR1	RO	0x02212000	Instruction Set Attributes Register 1
0xE000ED68	ID_ISAR2	RO	0x20232232	Instruction Set Attributes Register 2
0xE000ED6C	ID_ISAR3	RO	0x01111131	Instruction Set Attributes Register 3
0xE000ED70	ID_ISAR4	RO	0x01310132	Instruction Set Attributes Register 4
0xE000ED74	ID_ISAR5	RO	0x00000000	Instruction Set Attributes Register 5
0xE000ED78	CLIDR	RO	0xFFFF000X <b>Note:</b> Bits CLIDR[31:21] and CLIDR[2:0] depend on the cache configuration of the processor.	CLIDR, Cache Level ID Register
0xE000ED7C	CTR	RO	<ul style="list-style-type: none"><li>If an instruction cache or data cache is included, then the reset value is 0x8303C003.</li><li>If an instruction cache or data cache is not included, then the reset value is 0x00000000.</li></ul>	Cache Type Register



Address	Name	Type	Reset value	Description
0xE000ED80	CCSIDR	RO	0xFFFFFFFF <b>Note:</b> CCSIDR depends on the CSSELR setting and L1 cache configuration.	CCSIDR, Current Cache Size ID Register
0xE000ED84	CSSELR	RW	0x00000000	CSSELR, Cache Size Selection Register
0xE000EF40	MVFR0	RO	MVFR0, MVFR1, and MVFR2 reset values	Media and VFP Feature Register 0
0xE000EF44	MVFR1	RO		Media and VFP Feature Register 1
0xE000EF48	MVFR2	RO		Media and VFP Feature Register 2
0xE000EFD0	DPIDR4	RO	0x0000000A	CoreSight™ Peripheral ID Register 4
0xE000EFD4	DPIDR5	RO	0x00000000	CoreSight™ Peripheral ID Register 5
0xE000EFD8	DPIDR6	RO	0x00000000	CoreSight™ Peripheral ID Register 6
0xE000EFDC	DPIDR7	RO	0x00000000	CoreSight™ Peripheral ID Register 7
0xE000EFE0	DPIDR0	RO	0x00000024	CoreSight™ Peripheral ID Register 0
0xE000EFE4	DPIDR1	RO	0x0000005D	CoreSight™ Peripheral ID Register 1
0xE000EFE8	DPIDR2	RO	0x0000000F	CoreSight™ Peripheral ID Register 2

Address	Name	Type	Reset value	Description
0xE000EDEC	DPIDR3	RO	0x00000000 <b>Note:</b> Bits [7:4] and [3:0] are REVAND and CMOD respectively. The REVAND field indicates minor errata fixes specific to this design, for example metal fixes after implementation. If the component is reusable IP, the CMOD field indicates whether you have modified the behavior of the component. These values depend on the exact revision of the silicon as documented in <i>Arm® CoreSight™ Architecture Specification v3.0</i> .	CoreSight™ Peripheral ID Register 3
0xE000EFF0	DCIDR0	RO	0x0000000D	CoreSight™ Component ID Register 0
0xE000EFF4	DCIDR1	RO	0x00000090	CoreSight™ Component ID Register 1
0xE000EFF8	DCIDR2	RO	0x00000005	CoreSight™ Component ID Register 2
0xE000EDEC	DCIDR3	RO	0x000000B1	CoreSight™ Component ID Register 3
0xE000EFBC	DDEVARCH	RO	0x47702A04	CoreSight™ Device Architecture Register
0xE000EFCC	DDEVTYPE	RO	0x00000000	CoreSight™ Device Type Identifier Register
0xE000ECFC	REVIDR	RO	0x00000000 <b>Note:</b> The value of REVIDR[3:0] is determined by the input signal REVIDRNUM as specified in <a href="#">Miscellaneous signals</a> .	<a href="#">REVIDR, Revision ID Register</a>
0xE0005FC8	ERRDEVID	RO	0x00000001 <b>Note:</b> ERRDEVID[15:0] indicates the number of error records that the RAS Extension implementation supports. In the Cortex®-M52 processor, this field reads 0x0001 indicating one error record is supported.  This register is RAZ if any of the following conditions are true: <ul style="list-style-type: none"> <li>ECC protection is not configured.</li> <li>ECC protection is configured but not enabled</li> </ul>	<a href="#">ERRDEVID, RAS Error Record Device ID Register</a> .

## 5.2.1 Media and VFP Feature Register reset values, MVFR0, MVFR1, and MVFR2 reset values

The MVFR0, MVFR1, and MVFR2 register reset values depend on the *M-profile Vector Extension* (MVE) and floating-point functionality configuration. The MVE and floating-point functionality operation is configured using the `FPMVE` configuration parameters.

For more information, see [Cortex-M52 implementation options](#).

The following table shows the MVFR0, MVFR1, and MVFR2 reset values based on the reset configurations.

**Table 5-3: MVFR0, MVFR1, and MVFR2 reset values**

Configuration	MVFR0	MVFR1	MVFR2
FPMVE=0	0x00000000	0x00000000	0x00000000
FPMVE=1	0x10110021	0x11100011	0x00000040
FPMVE=2	0x10110221	0x12100011	0x00000040
FPMVE=3	0x00000001	0x00000100	0x00000000
FPMVE=4	0x10110021	0x11100111	0x00000040
FPMVE=5	0x10110221	0x12100211	0x00000040

## 5.3 AFSR, Auxiliary Fault Status Register

The AFSR provides fault status information.

### Usage constraints

Privileged access permitted only. Unprivileged accesses generate a fault. The register is set to zero at reset. A field in the register can be cleared by writing 0b1 to the corresponding bit. AFSR bits [31:21] are only valid if BFSR.IBUSERR is set. AFSR bits [20:10] are only valid if BFSR.PRECISEERR is set. AFSR bits [9:0] are only valid if BFSR.IMPRESISEERR is set. If multiple faults occur, the AFSR indicates the types of all the faults that have occurred. For more information on BFSR, see the *Arm®v8-M Architecture Reference Manual*.

If the Security Extension is implemented and AIRCR.BFHFNMINS is zero, this register is RAZ/WI from Non-secure state. Unprivileged access results in a BusFault exception

### Configuration

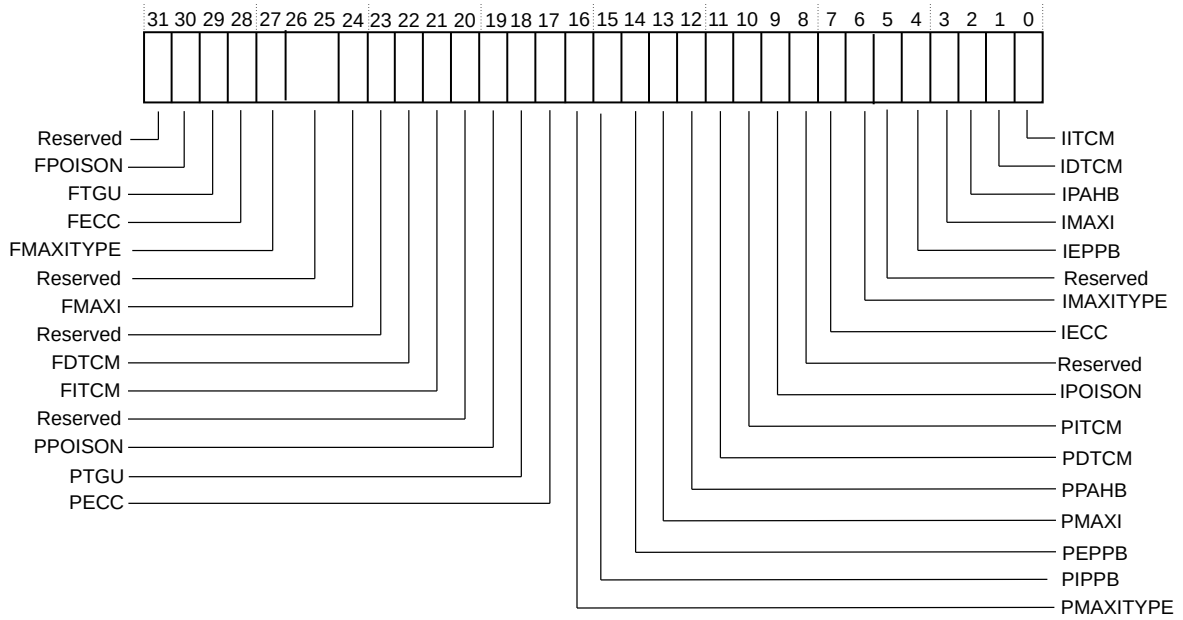
This register is always implemented.

### Attributes

A 32-bit RW register that is located at 0xE00ED3C. Non-secure alias is provided using AFSR\_NS, that is located at 0xE002ED3C. This register is not banked between Security states. See [IMPLEMENTATION DEFINED registers summary](#) for more information.

The following figure shows the AFSR bit assignments.

**Figure 5-1: AFSR bit assignments**



The following table describes the AFSR bit assignments.

**Table 5-4: AFSR bit assignments**

Bits	Name	Type	Description
[31]	Reserved	-	<b>RES0</b>
[30]	FPOISON	RW	Fetch fault that is caused by RPOISON or TEBRx.POISON.
[29]	FTGU	-	Fetch fault that is caused by <i>TCM Gate Unit</i> (TGU) security violation.
[28]	FECC	RW	Fetch fault that is caused by uncorrectable <i>Error Correcting Code</i> (ECC) error.
[27]	FMAXITYPE	RW	AXI response that caused the fetch fault. Only valid when AFSR.FMAXI is 1.  For M-AXI:  <b>0b0</b> SLVERR <b>0b1</b> DECERR  For M-AHB: <b>RES0</b>
[26:25]	Reserved	-	<b>RES0</b>
[24]	FMAXI	RW	Fetch fault on Main interface.
[23]	Reserved	-	<b>RES0</b>
[22]	FDTCM	RW	Fetch fault on <i>Data Tightly Coupled Memory</i> (DTCM) interface.
[21]	FITCM	RW	Fetch fault on <i>Instruction Tightly Coupled Memory</i> (ITCM) interface.
[20]	Reserved	-	<b>RES0</b>
[19]	PPOISON	RW	Precise fault that is caused by RPOISON of M-AXI or TEBRx.POISON.
[18]	PTGU	RW	Precise fault that is caused by TGU security violation.

Bits	Name	Type	Description
[17]	PECC	RW	Precise fault that is caused by uncorrectable ECC error.
[16]	PMAXITYPE	RW	AXI response that caused the precise fault. Only valid when AFSR.PMAXI is 1.  For M-AXI:  <div> <div>0b0</div> <div>0b1</div> <div>SLVERR</div> <div>DECERR</div> </div> For M-AHB: <b>RES0</b>
[15]	PIPPB	RW	Precise fault on <i>Internal Private Peripheral Bus</i> (IPPB) interface.
[14]	PEPPB	RW	Precise fault on <i>External Private Peripheral Bus</i> (EPPB) interface.
[13]	PMAXI	RW	Precise fault on Main interface.
[12]	PPAHB	RW	Precise fault on <i>Peripheral AHB</i> (P-AHB) interface.
[11]	PDTCM	RW	Precise fault on DTCM interface.
[10]	PITCM	RW	Precise fault on ITCM interface.
[9]	IPOISON	RW	Imprecise BusFault because of RPOISON.
[8]	Reserved	-	<b>RES0</b>
[7]	IECC	RW	Imprecise fault that is caused by uncorrectable ECC error.
[6]	IMAXITYPE	RW	AXI response that caused the imprecise fault. Only valid when AFSR.IMAXI is 1.  For M-AXI:  <div> <div>0b0</div> <div>0b1</div> <div>SLVERR</div> <div>DECERR</div> </div> For M-AHB: <b>RES0</b>
[5]	Reserved	-	<b>RES0</b>
[4]	IEPPB	RW	Imprecise fault on EPPB interface.
[3]	IMAXI	RW	Imprecise fault on Main interface.
[2]	IPAHB	RW	Imprecise fault on P-AHB interface.
[1]	IDTCM	RW	Imprecise fault on DTCM interface.
[0]	IITCM	RW	Imprecise fault on ITCM interface.

## 5.4 CPUID, CPUID Base Register

CPUID contains the Cortex®-M52 processor part number, version, and implementation information.

### Usage constraints

This register is read-only.

### Configuration

This register is always implemented.

## Attributes

This register is not banked between Security states. See [Identification register summary](#) for more information.

The following figure shows the CPUID bit assignments.

**Figure 5-2: CPUID bit assignments**

31	24:23	20:19	16:15	4	3	0
Implementer	Variant	Architecture	PartNo	Revision		

The following table shows the CPUID bit assignments.

**Table 5-5: CPUID bit assignments**

Bits	Name	Type	Description
[31:24]	Implementer	RO	Implementer code that Arm China has assigned.  <b>0x63</b> c: Arm China
[23:20]	Variant	RO	Variant number to distinguish between different product variants or major revisions of the product. Variant is the x in the rxxpy product revision identifier.  <b>0x0</b> Cortex®-M52 r0p2
[19:16]	Architecture	RO	Indicates the architecture version that the Cortex®-M52 processor implements.  <b>0b1111</b> Arm®v8.1-M with Main Extension.
[15:4]	PartNo	RO	Part number of the Cortex®-M52 processor.  <b>0xD24</b> Cortex®-M52
[3:0]	Revision	RO	Revision number to distinguish between different patches of the product. Revision is the y in the rxxpy product revision identifier.  <b>0x2</b> Cortex®-M52 r0p2

## 5.5 ID\_AFR0, Auxiliary Feature Register 0

The ID\_AFR0 register provides information about the **IMPLEMENTATION DEFINED** features of the processor.

### Usage constraints

Privileged access permitted only. Unprivileged accesses generate a fault.

This register is word accessible only. Halfword and byte accesses are **UNPREDICTABLE**.

## Configurations

This register is always implemented.

## Attributes

This is a 32-bit read-only register.

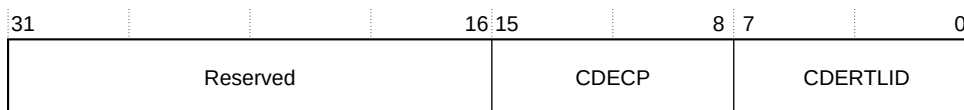
ID\_AFR0\_S is located at 0xE000ED4C.

ID\_AFR0\_NS is located at 0xE002ED4C. ID\_AFR0\_NS is **RES0** to software executing in Non-secure state and the debugger.

This register is not banked between Security states.

The following figure shows the ID\_AFR0 bit assignments.

**Figure 5-3: ID\_AFR0 bit assignments**



The following table shows the ID\_AFR0 bit assignments.

**Table 5-6: ID\_AFR0 bit assignments**

Bits	Name	Type	Function
[31:16]	-	-	Reserved, <b>RES0</b>
[15:8]	CDECP	RO	For each coprocessor, this field indicates whether the coprocessor is used by a CDE module and not by the coprocessor interface. The values can be:  <b>0</b> Coprocessor used by the coprocessor interface. <b>1</b> Coprocessor used by a CDE module.
[7:0]	CDERTLID	RO	Software can use this field to read the value of the <b>CDERTLID</b> parameter. This parameter manages the CDE customization that might be needed in systems with more than one Cortex®-M52 processor.

## 5.6 Cache identification register summary

The cache identification registers are responsible for cache configuration in the processor. The fields in these registers depend on the instruction and data cache size.

The following table lists the cache identification registers.





**Table 5-8: CLIDR bit assignments**

Bits	Name	Type	Description
[31:30]	ICB	RO	Inner cache boundary. The Cortex®-M52 processor supports inner Cacheability on the bus. Therefore, this field cannot disclose any information.  <b>0b00</b> Not disclosed in this mechanism.
[29:27]	LoUU	RO	Level of Unification Uniprocessor. The L1 cache must be cleaned or invalidated when cleaning or invalidating occurs to the point of unification. The options are:  <b>0b000</b> Caches are not implemented. Therefore, cleaning and invalidation is not required. <b>0b001</b> Level 1 (L1) data cache or instruction cache is implemented. Therefore, cleaning and invalidation are required.
[26:24]	LoC	RO	Level of Coherency. The L1 cache must be cleaned when cleaning occurs to the point of coherency. The options are:  <b>0b000</b> Caches are not implemented. Therefore, cleaning is not required. <b>0b001</b> L1 data cache or instruction cache is implemented. Therefore, cleaning is required.
[23:21]	LoUIS	RO	Level of Unification Inner Shareable. The L1 cache must be cleaned or invalidated when cleaning or invalidating occurs to the point of unification for the inner Shareability domain. The options are:  <b>0b000</b> Caches are not implemented. Therefore, cleaning and invalidation are not required. <b>0b001</b> L1 data cache or instruction cache is implemented. Therefore, cleaning and invalidation are required.
[20:3]	Reserved	-	<b>RES0</b>
[2:0]	Ctype1	RO	Level 1 (L1) cache type. The options are:  <b>0b000</b> Caches are not implemented. <b>0b001</b> Only instruction cache is implemented. <b>0b010</b> Only data cache is implemented. <b>0b011</b> Both data cache and instruction cache are implemented. <b>0b100</b> Unified cache.

## 5.6.2 CSSELR, Cache Size Selection Register

The CSSELR selects the cache accessed through the CCSIDR by specifying the cache level and the type of cache (either instruction or data cache). For Cortex®-M52, this can be either the L1 instruction cache or L1 data cache.

### Usage constraints

This register is read/write and is accessible in Privileged mode only.

### Configurations

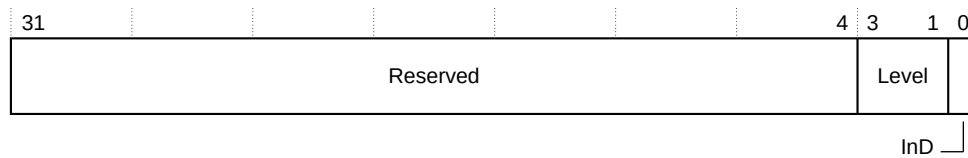
This register is always implemented.

### Attributes

See [Identification register summary](#) for more information.

This register is banked between Security states. The following figure shows the CSSELR bit assignments.

**Figure 5-5: CSSELR bit assignments**



The following table shows the CSSELR bit assignments.

**Table 5-9: CSSELR bit assignments**

Bits	Name	Type	Function
[31:4]	Reserved	-	<b>RES0</b>
[3:1]	Level	RO	Identifies which cache level to select.  <b>0x0</b> L1 cache.  This field is RAZ/WI.
[0]	InD	RW	Selects either L1 instruction or data cache. The options are:  <b>0</b> L1 data cache, or L1 unified cache. <b>1</b> L1 instruction cache.

### 5.6.3 CCSIDR, Current Cache Size ID Register

The CCSIDR provides information about the architecture of the instruction or data cache that the CSSELR selects. If the cache corresponding to CSSELR.InD is not included in the processor, then this register reads 0x00000000.

#### Usage constraints

This register is read-only and is accessible in Privileged mode only.

#### Configurations

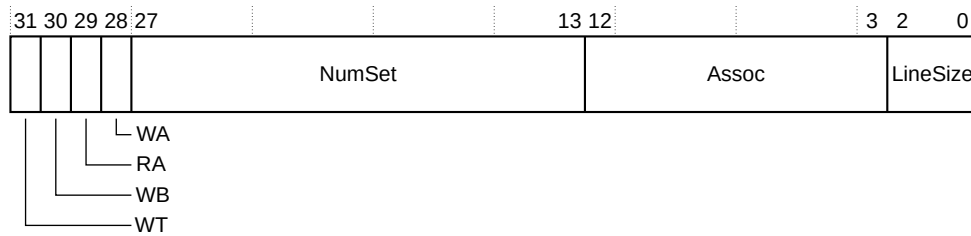
This register is always implemented.

#### Attributes

This register is banked between Security states. The value of this register depends on the cache that CSSELR selects. If you are setting CSSELR in a particular Security state, then Arm® recommends that you read CCSIDR in the same Security state to get the architecture information about the selected instruction or data cache.

The following figure shows the CCSIDR bit assignments.

**Figure 5-6: CCSIDR bit assignments**



The following table shows the CCSIDR bit assignments.

**Table 5-10: CCSIDR bit assignments**

Bits	Name	Type	Function
[31]	WT	RO	Indicates support available for Write-Through:  <b>0b1</b> Write-Through support available.
[30]	WB	RO	Indicates support available for Write-Back:  <b>0b1</b> Write-Back support available.
[29]	RA	RO	Indicates support available for read allocation:  <b>0b1</b> Read allocation support available.
[28]	WA	RO	Indicates support available for write allocation:  <b>0b1</b> Write allocation support available.
[27:13]	NumSet	RO	Indicates the number of sets.  Cache-size dependent.
[12:3]	Assoc	RO	Indicates associativity. The value depends on the cache that CSSELR selects.  <b>0x1</b> 2-way set associative cache (instruction or unified cache).  <b>0x3</b> 4-way set associative cache (data cache).
[2:0]	LineSize	RO	Indicates the number of words in each cache line.  <b>0b1</b> Represents 32 bytes.

The LineSize field is encoded as 2 less than  $\log_2$  of the number of words in the cache line. For example, a value of **0x0** indicates that there are four words in a cache line, that is the minimum size for the cache. A value of **0x1** indicates that there are eight words in a cache line.

## 5.7 REVIDR, Revision ID Register

The REVIDR register provides additional **IMPLEMENTATION-SPECIFIC** minor revision that can be interpreted with the CPUID register.

### Usage constraints

- Unprivileged access results in a BusFault exception. If the Security Extension is implemented, this register is RAZ/WI from Non-secure state.
- This register is accessible through unprivileged *Debug AHB* (D-AHB) debug requests when either DAUTHCTRL\_S.UIDAPEN or DAUTHCTRL\_NS.UIDAPEN is set.

### Configurations

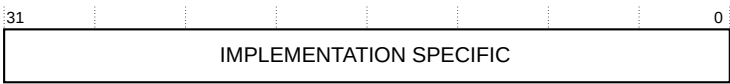
- This register is always implemented.

### Attributes

- This register is not banked between Security states. See [IMPLEMENTATION DEFINED registers summary](#) for more information.

The following figure shows the REVIDR bit assignments.

Figure 5-7: REVIDR bit assignments



The following table describes the REVIDR bit assignments.

Table 5-11: REVIDR bit assignments

Field	Name	Type	Description
[31:0]	IMPLEMENTATION SPECIFIC	RO	<b>IMPLEMENTATION-SPECIFIC</b> minor revision information that can be interpreted with the CPUID register. For more information on the CPUID register, see the <i>Arm®v8-M Architecture Reference Manual</i> .



The value of REVIDR[3:0] is determined by the input signal REVIDRNUM as specified in [Miscellaneous signals](#)

## 5.8 Implementation control register summary

Implementation control registers are architecturally defined with values that control aspects of system implementation.

The following table shows a summary of the implementation control registers. For more information on the architectural registers that are listed in the following table, see the *Arm®v8-M Architecture Reference Manual*.

**Table 5-12: Implementation control register summary**

Address	Name	Type	Reset value	Description
0xE000E004	ICTR	RO	0x0000000X <b>Note:</b> ICTR[3:0] depends on the number of interrupts that are included in the processor. Bits [31:4] are zero.	ICTR, Interrupt Controller Type Register
0xE000E008	ACTLR	RW	0x00000000	ACTLR, Auxiliary Control Register
0xE000E00C	CPPWR	RW	0x00000000	Coprocessor Power Control Register

## 5.9 ACTLR, Auxiliary Control Register

The ACTLR contains many fields that allow software to control the processor features and functionality.

### Usage constraints

Privileged access permitted only. Unprivileged accesses generate a BusFault exception.

### Configuration

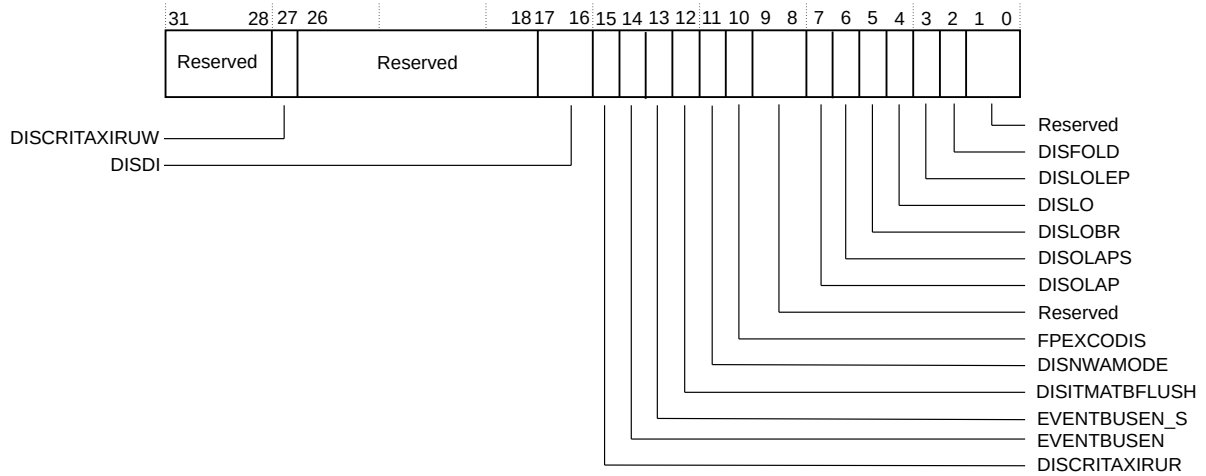
This register is always implemented.

### Attributes

A 32-bit RW register that is located at 0xE000E008. Non-secure alias is provided using ACTLR\_NS, located at 0xE002E008. This register is banked between Security domains. See [IMPLEMENTATION DEFINED registers summary](#) for more information. At reset, all fields in this register are set to zero.

The following figure shows the ACTLR bit assignments.

**Figure 5-8: ACTLR bit assignments**



The following table describes the ACTLR bit assignments.

**Table 5-13: ACTLR bit assignments**

Bits	Name	Type	Description
[31:28]	Reserved	-	These bits are reserved for future use and must be treated as UNK/SBZP.
[27]	DISCRITAXIRUW	RW	<p>Disable-Critical-AXI-Read-Under-Write. The options are:</p> <p><b>0</b> Normal operation.</p> <p><b>1</b> AXI reads to Device memory and exclusive reads to shared memory are not initiated on the M-AXI read address channel until all outstanding writes on the M-AXI interface are complete.</p> <p>Setting this bit decreases performance.</p>
[26:18]	Reserved	-	These bits are reserved for future use and must be treated as UNK/SBZP.
[17:16]	DISDI	RW	<p>Disable dual-issue features. The options for this bit are:</p> <p><b>0b00</b> Full dual-issue, if DISFOLD is set to 0.</p> <p><b>0b01</b> Disable dual-issue of arithmetic instructions.</p> <p>Other values are reserved.</p>
[15]	DISCRITAXIRUR	RW	<p>Disable critical AXI Read-Under-Read. The options for this bit are:</p> <p><b>0</b> Normal operation.</p> <p><b>1</b> AXI reads to Device memory and exclusive reads to shared memory are not initiated on the M-AXI read address channels if there are any outstanding reads on the M-AXI. Transactions on the M-AXI cannot be interrupted.</p> <p>This bit might reduce the time that these transactions are in progress and might improve worst-case interrupt latency. Setting this bit reduces performance.</p>

Bits	Name	Type	Description
[14]	EVENTBUSEN	RW	<p>Activate EVENTBUS output</p> <p><b>0</b> EVENTBUS not active <b>1</b> EVENTBUS active</p> <p>This bit resets to 0 on Warm reset, and this bit is not banked.</p>
[13]	EVENTBUSEN_S	RW	<p>Accessibility of EVENTBUSEN</p> <p><b>0</b> EVENTBUSEN is accessible by both Security states <b>1</b> EVENTBUSEN is accessible by Secure state only.</p> <p>This bit is RAZ/WI from Non-secure state. This bit resets to 0 on Warm reset.</p>
[12]	DISITMATBFLUSH	RW	<p>This bit determines whether <i>Instrumentation Trace Macrocell</i> (ITM) or <i>Data Watchpoint and Trace</i> (DWT) ATB flush is disabled. The options for this bit are:</p> <p><b>0</b> Normal operation. <b>1</b> ITM or DWT ATB flush is disabled.</p> <p>When disabled, the AFVALID signal (trace flush request) is ignored and the AFREADY (trace flush ready) signal is held HIGH. This field only resets on Cold reset.</p>
[11]	DISNWAMODE	RW	<p>This bit determines if no write allocate mode is disabled. The options for this bit are:</p> <p><b>0</b> Normal operation. <b>1</b> No write allocate mode is disabled.</p> <p>Setting this bit decreases performance. For more information on no write allocation mode, see <a href="#">No Write-Allocate mode</a>.</p>
[10]	FPEXCODIS	RW	<p>This bit determines if floating-point exception outputs are disabled. The options for this bit are:</p> <p><b>0</b> Normal operation. <b>1</b> Floating-point exception outputs are disabled.</p>
[9:8]	Reserved	-	These bits are reserved for future use and must be treated as UNK/SBZP.
[7]	DISOLAP	RW	Disable overlapping of all instructions.
[6]	DISOLAPS	RW	Disable overlapping of scalar-only instructions.
[5]	DISLOBR	RW	<p>Disable branch prediction using low overhead loops.</p> <p><b>0</b> Branch prediction enabled <b>1</b> Branch prediction disabled.</p> <p>This field is reset to 0b0. If DISLO is set, then branch prediction is disabled regardless of this bit.</p>
[4]	DISLO	RW	<p>Disable low overhead loops. The options are:</p> <p><b>0</b> Low overhead loops enabled. <b>1</b> Low overhead loops disabled.</p>

Bits	Name	Type	Description
[3]	DISLOLEP	RW	Disable end of loop prediction in low overhead loops. The options are:  <b>0</b> Low overhead loop end prediction enabled <b>1</b> Low overhead loop end prediction disabled  Setting this bit decreases performance.
[2]	DISFOLD	RW	This bit determines if dual-issue functionality is disabled. The options are:  <b>0</b> Normal operation. <b>1</b> Dual-issue functionality is disabled.  Setting this bit decreases performance.
[1:0]	Reserved	-	These bits are reserved for future use and must be treated as UNK/SBZP.

## 5.10 ICTR, Interrupt Controller Type Register

The ICTR register shows the number of interrupt lines that the NVIC supports.

### Usage Constraints

There are no usage constraints.

### Configurations

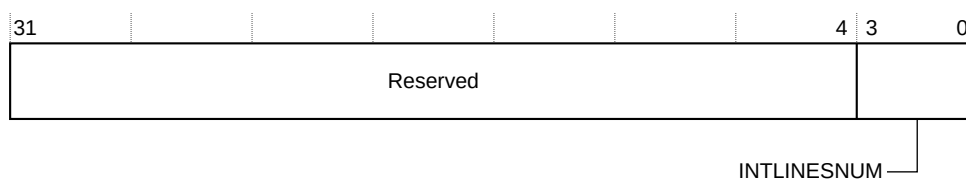
This register is available in all processor configurations.

### Attributes

See [NVIC register summary](#) for more information.

The following figure shows the ICTR bit assignments.

**Figure 5-9: ICTR bit assignments**



The following table shows the ICTR bit assignments.

**Table 5-14: ICTR bit assignments**

Bits	Name	Type	Function
[31:4]	-	-	Reserved.



Bits	Name	Type	Function
[3:0]	INTLINESNUM	RO	Total number of interrupt lines in groups of 32:  <div> <div>0b0000</div> <div>0b0001</div> <div>0b0010</div> <div>0b0011</div> <div>0b0100</div> <div>0b0101</div> <div>0b0110</div> <div>0b0111</div> <div>0b1000</div> <div>0b1001</div> <div>0b1010</div> <div>0b1011</div> <div>0b1100</div> <div>0b1101</div> <div>0b1110</div> </div> <div> <div>0-32</div> <div>33-64</div> <div>65-96</div> <div>97-128</div> <div>129-160</div> <div>161-192</div> <div>193-224</div> <div>225-256</div> <div>257-288</div> <div>289-320</div> <div>321-352</div> <div>353-384</div> <div>385-416</div> <div>417-448</div> <div>449-480</div> </div>



The processor supports a maximum of 480 external interrupts.

## 5.11 IMPLEMENTATION DEFINED registers summary

The 32-bit **IMPLEMENTATION DEFINED** registers provide memory configuration and access control, error record information, interrupt control, and processor configuration information.

The following table lists the **IMPLEMENTATION DEFINED** registers for the Cortex®-M52 processor.

**Table 5-15: IMPLEMENTATION DEFINED registers summary**

Address	Name	Type	Reset value	Description
0xE0005000	ERRFRO	RO	0x00000101	ERRFRO, RAS Error Record Feature Register
0xE0005008	ERRCTRL0	-	-	This register is <b>RES0</b> .
0xE0005010	ERRSTATUS0	RW	0xFFFF000X	ERRSTATUS0, RAS Error Record Primary Status Register
0xE0005018	ERRADDR0	RW	0xFFFFFFFF	ERRADDR0 and ERRADDR20, RAS Error Record Address Registers
0xE000501C	ERRADDR20	RO	0x00000000	ERRADDR0 and ERRADDR20, RAS Error Record Address Registers
0xE0005020	ERRMISC00	-	-	This register is <b>RES0</b> .
0xE0005024	ERRMISC10	RO	0x0000000X	ERRMISC10, Error Record Miscellaneous Register 10
0xE0005028	ERRMISC20	-	-	This register is <b>RES0</b> .
0xE000502C	ERRMISC30	-	-	This register is <b>RES0</b> .

Address	Name	Type	Reset value	Description
0xE0005030	ERRMISC40	-	-	This register is <b>RES0</b> .
0xE0005034	ERRMISC50	-	-	This register is <b>RES0</b> .
0xE0005038	ERRMISC60	-	-	This register is <b>RES0</b> .
0xE000503C	ERRMISC70	-	-	This register is <b>RES0</b> .
0xE0005E00	ERRGSRO	RO	0x00000000	ERRGSRO, RAS Fault Group Status Register
0xE000ECFC	REVIDR	RO	0x00000000	REVIDR, Revision ID Register
0xE0005FC8	ERRDEVID	RO	0x00000001	ERRDEVID, RAS Error Record Device ID Register
0xE000E008	ACTLR	RW	0x00000000	ACTLR, Auxiliary Control Register
0xE000ED3C	AFSR	RW	0x00000000	AFSR, Auxiliary Fault Status Register
0xE000EF04	RFSR	RW	0XXXX000X	RFSR, RAS Fault Status Register
0xE001E000	MSCR	RW	If the instruction cache and data cache are not present, then the reset value is 0x0000000X.  If the instruction cache and data cache are present, then the reset value is 0x0000300X.	MSCR, Memory System Control Register
0xE001E010	ITCMCR	RW	0x000000XX	ITCMCR and DTCMCR, TCM Control Registers
0xE001E014	DTCMCR	RW	0x000000XX	
0xE001E018	PAHBCR	RW	0x0000000X.	PAHBCR, P-AHB Control Register
0xE001E100	IEBR0	RW	0x00000000	IEBR0 and IEBR1, Instruction Cache Error Bank Register 0-1
0xE001E104	IEBR1	RW	0x00000000	
0xE001E110	DEBR0	RW	0x00000000	DEBR0 and DEBR1, Data Cache Error Bank Register 0-1
0xE001E114	DEBR1	RW	0x00000000	
0xE001E120	TEBR0	RW	0x00000000	TEBR0 and TEBR1, TCM Error Bank Register 0-1
0xE001E124	TEBRDATA0	RO	0x00000000	Data for TCU Error Bank Register 0-1, TEBRDATA0 and TEBRDATA1
0xE001E128	TEBR1	RW	0x00000000	TEBR0 and TEBR1, TCM Error Bank Register 0-1
0xE001E12C	TEBRDATA1	RO	0x00000000	Data for TCU Error Bank Register 0-1, TEBRDATA0 and TEBRDATA1
0xE001E200	DCADCRR	RO	DCAICRR and DCADCRR, Direct Cache Access Read Registers	
0xE001E204	DCAICRR	RO		
0xE001E210	DCADCLR	RW	0x00000000	
0xE001E214	DCAICLR	RW	0x00000000	
0xE001E300	CPDLPSTATE	RW	0x00000303	CPDLPSTATE, Core Power Domain Low Power State Register
0xE001E304	DPDLPSTATE	RW	0x00000003	DPDLPSTATE, Debug Power Domain Low Power State Register
0xE001E400	EVENTSPR	WO	0x0000000X	EVENTSPR, Event Set Pending Register

Address	Name	Type	Reset value	Description
0xE001E480	EVENTMASKA	RO	0x0000000X	EVENTMASKA and EVENTMASKn, n=0-14, Wakeup Event Mask Registers
0xE001E484 4n	EVENTMASKn	RO	UNKNOWN	
0xE001E500	ITGU_CTRL	RW	0x00000003	ITGU_CTRL and DTGU_CTRL, ITGU and DTGU Control Registers
0xE001E504	ITGU_CFG	RO	0xX0002X0X	ITGU_CFG and DTGU_CFG, ITGU and DTGU Configuration Registers
0xE001E510 + 4n	ITGU_LUTn	RW if 32n +1<2 <sup>Number of ITGU blocks</sup>  RO if 32n +1≥2 <sup>Number of ITGU blocks</sup>	0x00000000	ITGU_LUTn and DTGU_LUTn, ITGU and DTGU Look Up Table Registers
0xE001E600	DTGU_CTRL	RW	0x00000003	ITGU_CTRL and DTGU_CTRL, ITGU and DTGU Control Registers
0xE001E604	DTGU_CFG	RO	0xX0002X0X	ITGU_CFG and DTGU_CFG, ITGU and DTGU Configuration Registers
0xE001E610 + 4n	DTGU_LUTn	RW if 32n +1<2 <sup>Number of DTGU blocks</sup>  RO if 32n +1≥2 <sup>Number of DTGU blocks</sup>	0x00000000	ITGU_LUTn and DTGU_LUTn, ITGU and DTGU Look Up Table Registers
0xE001E700	CFGINFOSEL	WO	UNKNOWN	CFGINFOSEL, Processor configuration information selection register
0xE001E704	CFGINFORD	RO	UNKNOWN	CFGINFORD, Processor configuration information read data register
0xE001E800	STLNVICPENDOR	RO	0x00000000	STLNVICPENDOR and STLNVICACTVOR, NVIC observation registers
0xE001E804	STLNVICACTVOR	RO	0x00000000	
0xE001E810	STLDMPUSR	RW	0x00000000	STLDMPUSR, STLIMPUOR, and STLDMPUOR, MPU observation registers
0xE001E814	STLIMPUOR	RO	0x00000000	
0xE001E818	STLDMPUOR	RO	0x00000000	

The following registers are reset on Cold reset only. These reset values persist across a system reset or Warm reset.



Note

- [ERRFR0](#), RAS Error Record Feature Register.
- [ERRMISC10](#), Error Record Miscellaneous Register 10.
- [ERRADDR0](#) and [ERRADDR20](#), RAS Error Record Address Registers.
- [ERRSTATUS0](#), RAS Error Record Primary Status Register.

- [ERRGSRO, RAS Fault Group Status Register](#).

## 5.12 Direct cache access registers

The Cortex®-M52 processor provides a set of **IMPLEMENTATION DEFINED** registers that allows direct read access to the embedded RAM associated with the L1 instruction and data cache. Two registers are included for each cache, one to set the required RAM and location, and the other to read out the data. L1 unified cache reuses the direct cache access registers of data cache.

The following table lists the direct cache access registers.

**Table 5-16: Direct cache access registers**

Address	Name	Type	Reset value	Description
0xE001E200	DCADCRR	RO	UNKNOWN	DCAICRR and DCADCRR, Direct Cache Access Read Registers
0xE001E204	DCAICRR	RO	UNKNOWN	
0xE001E210	DCADCLR	RW	0x00000000	DCAICLR and DCADCLR, Direct Cache Access Location Registers
0xE001E214	DCAICLR	RW	0x00000000	

### 5.12.1 DCAICLR and DCADCLR, Direct Cache Access Location Registers

The DCAICLR and DCADCLR registers are used by software to set the location to be read from the L1 instruction cache and data cache respectively. The unified cache reuses registers of data cache.

#### Usage Constraints

The DCAICLR is RAZ/WI if the L1 instruction cache is not present. The DCADCLR is RAZ/WI if the L1 data cache and L1 unified cache are not present. If the Security Extension is implemented, these registers are RAZ/WI from the Non-secure state. Unprivileged access results in a BusFault exception.

#### Configurations

These registers are always implemented.

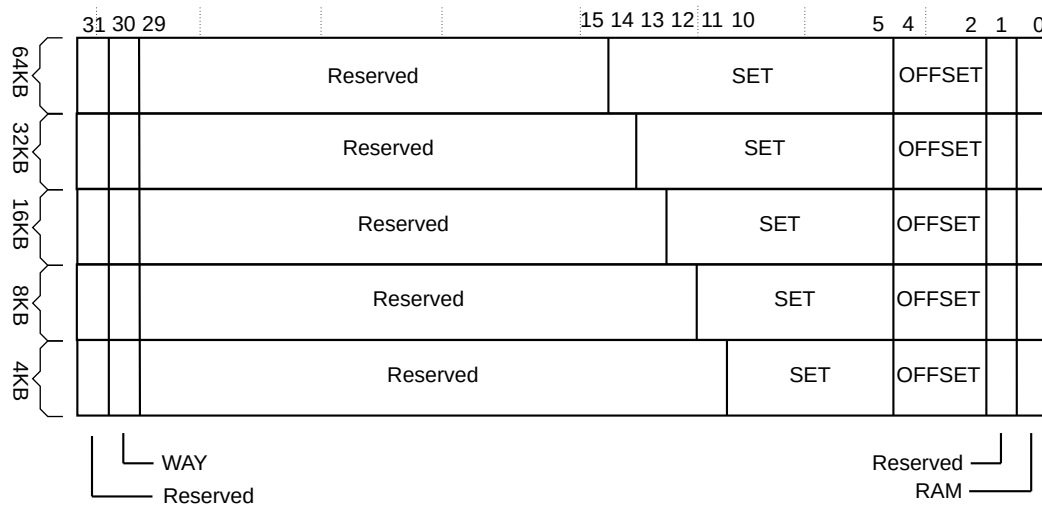
#### Attributes

These registers are not banked between Security states. See [IMPLEMENTATION DEFINED registers summary](#) for more information.

#### DCAICLR

The following figure shows the DCAICLR bit assignments.

**Figure 5-10: DCAICLR bit assignments**



The following table shows the DCAICLR bit assignments.

**Table 5-17: DCAICLR bit assignments**

Bits	Name	Type	Function
[31]	Reserved	-	RES0
[30]	WAY	RW	Cache way
[29:N+1]	Reserved	-	Set index. The value of N depends on the cache size.
[N:5]	SET	RW	The options are:  <div> <div>64KB</div> <div>32KB</div> <div>16KB</div> <div>8KB</div> <div>4KB</div> </div> <div> <div>N=14</div> <div>N=13</div> <div>N=12</div> <div>N=11</div> <div>N=10</div> </div>
[4:2]	OFFSET	RW	Data offset
[1]	Reserved	-	RES0
[0]	RAMTYPE	RW	RAM type  <div> <div>0</div> <div>1</div> </div> <div> <div>Tag RAM</div> <div>Data RAM</div> </div>

## DCADCLR

The following figures show the DCADCLR bit assignments.

Figure 5-11: DCADCLR bit assignments for data cache

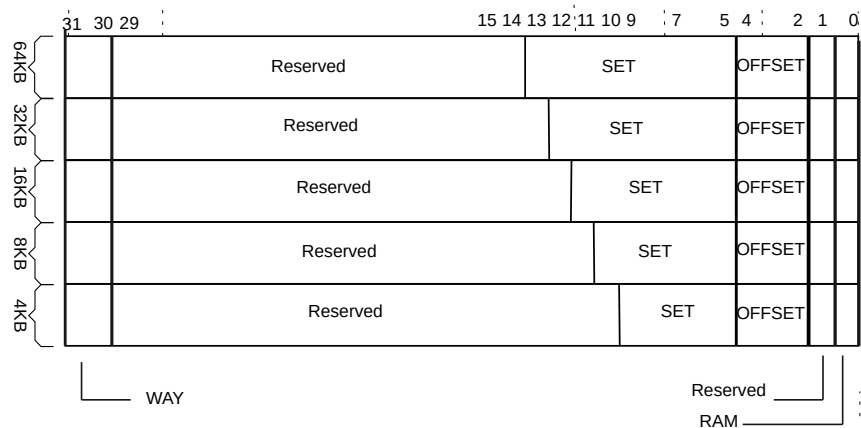
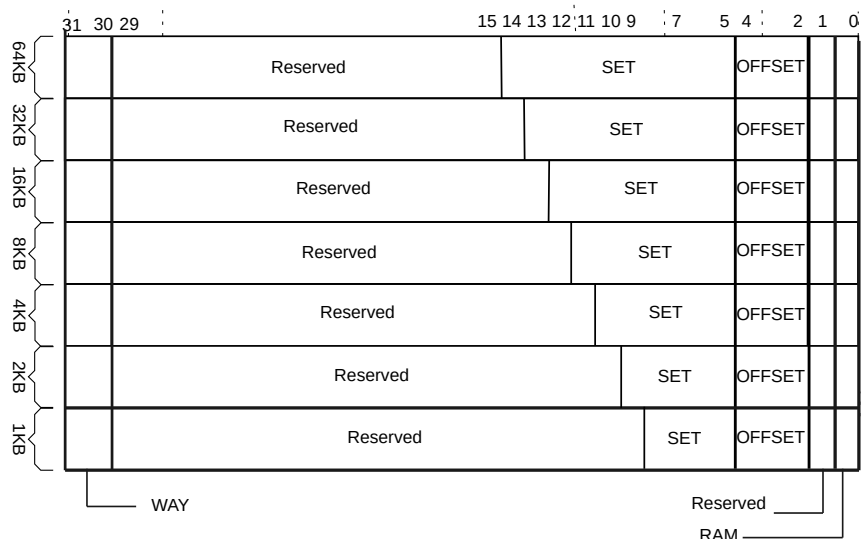


Figure 5-12: DCADCLR bit assignments for unified cache



The following table shows the DCADCLR bit assignments.

**Table 5-18: DCADCLR bit assignments**

Bits	Name	Type	Function																												
[31:30]	WAY	RW	Cache way Bit[31] is reserved when configured with unified cache.																												
[29:N+1]	Reserved	-	Set index. The value of N depends on the cache size.																												
[N:5]	SET	RW	For data cache, the options are:  <table><tr><td><b>64KB</b></td><td>N=13</td></tr><tr><td><b>32KB</b></td><td>N=12</td></tr><tr><td><b>16KB</b></td><td>N=11</td></tr><tr><td><b>8KB</b></td><td>N=10</td></tr><tr><td><b>4KB</b></td><td>N=9</td></tr><tr><td><b>2KB</b></td><td>NA</td></tr><tr><td><b>1KB</b></td><td>NA</td></tr></table> For unified cache, the options are:  <table><tr><td><b>64KB</b></td><td>N=14</td></tr><tr><td><b>32KB</b></td><td>N=13</td></tr><tr><td><b>16KB</b></td><td>N=12</td></tr><tr><td><b>8KB</b></td><td>N=11</td></tr><tr><td><b>4KB</b></td><td>N=10</td></tr><tr><td><b>2KB</b></td><td>N=9</td></tr><tr><td><b>1KB</b></td><td>N=8</td></tr></table> 1KB and 2KB are only applicable to unified cache where reserved bits are <b>RES0</b> .	<b>64KB</b>	N=13	<b>32KB</b>	N=12	<b>16KB</b>	N=11	<b>8KB</b>	N=10	<b>4KB</b>	N=9	<b>2KB</b>	NA	<b>1KB</b>	NA	<b>64KB</b>	N=14	<b>32KB</b>	N=13	<b>16KB</b>	N=12	<b>8KB</b>	N=11	<b>4KB</b>	N=10	<b>2KB</b>	N=9	<b>1KB</b>	N=8
<b>64KB</b>	N=13																														
<b>32KB</b>	N=12																														
<b>16KB</b>	N=11																														
<b>8KB</b>	N=10																														
<b>4KB</b>	N=9																														
<b>2KB</b>	NA																														
<b>1KB</b>	NA																														
<b>64KB</b>	N=14																														
<b>32KB</b>	N=13																														
<b>16KB</b>	N=12																														
<b>8KB</b>	N=11																														
<b>4KB</b>	N=10																														
<b>2KB</b>	N=9																														
<b>1KB</b>	N=8																														
[4:2]	OFFSET	RW	Data offset																												
[1]	Reserved	-	<b>RES0</b>																												
[0]	RAMTYPE	RW	RAM type  <table><tr><td><b>0</b></td><td>Tag RAM</td></tr><tr><td><b>1</b></td><td>Data RAM</td></tr></table>	<b>0</b>	Tag RAM	<b>1</b>	Data RAM																								
<b>0</b>	Tag RAM																														
<b>1</b>	Data RAM																														

## 5.12.2 DCAICRR and DCADCRR, Direct Cache Access Read Registers

The *Direct Cache Access Instruction Cache Read Register* (DCAICRR) and *Direct Cache Access Data Cache Read Register* (DCADCRR) registers are used by software to read the data from the L1 instruction cache, data cache or unified cache from the location that the DCAICLR and DCADCLR registers determine. The unified cache reuses registers of data cache.

### Usage Constraints

The DCAICRR is RAZ if the L1 instruction cache is not present. The DCADCRR is RAZ if the L1 data cache and L1 unified cache are not present.

If the Security Extension is implemented, then this register is RAZ from the Non-secure state. Unprivileged access results in a BusFault exception.

These registers are also RAZ/WI if any of the following conditions are true:

- MSCR.ICAACTIVE or MSCR.DCAACTIVE is 0.

- PDRAMS is not powered up and clocked.
- The instruction or data cache is being automatically invalidated.

## Configurations

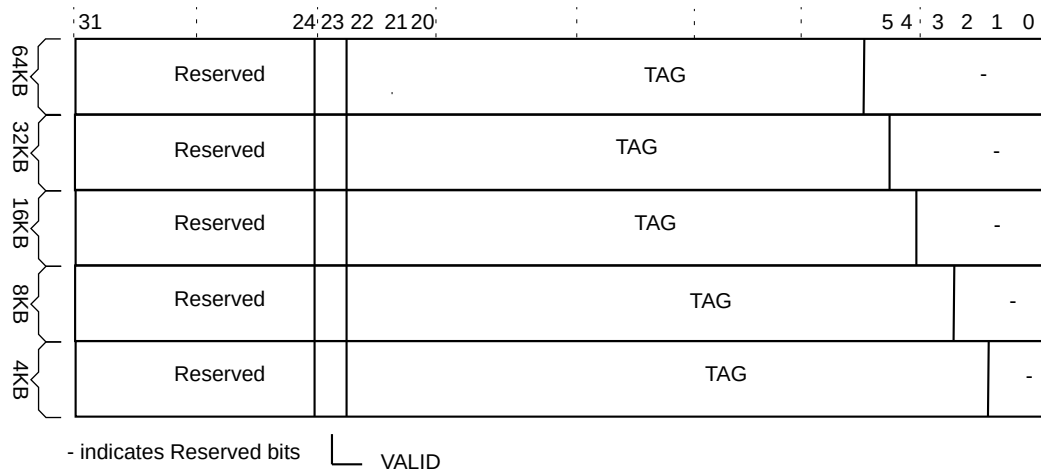
These registers are always implemented.

## Attributes

These registers are read-only and ignore all writes. These registers are not banked between Security states. See [IMPLEMENTATION DEFINED registers summary](#) for more information.

The following figure shows the DCAICRR bit assignments when reading the instruction cache tag RAM.

**Figure 5-13: DCAICRR bit assignments when reading the instruction cache tag RAM**



The following table shows the DCAICRR bit assignments when reading the instruction cache tag RAM.

**Table 5-19: DCAICRR bit assignments when reading the instruction cache tag RAM**

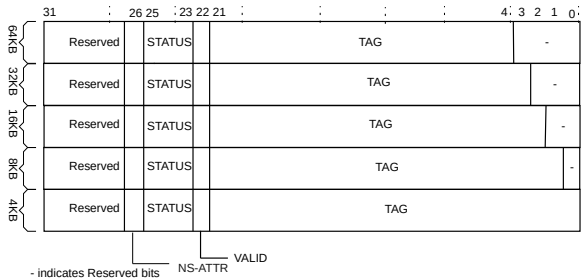
Bits	Name	Type	Function
[31:24]	-	-	RES0
[23]	VALID	RO	Valid state of the instruction cache line.
[22:N]	TAG	RO	Tag address. The number of significant bits of TAG depends on the instruction cache size.  <div> <div>64KB</div> <div>32KB</div> <div>16KB</div> <div>8KB</div> <div>4KB</div> </div> <div> <div>N=6</div> <div>N=5</div> <div>N=4</div> <div>N=3</div> <div>N=2</div> </div>



Bits	Name	Type	Function
[N-1:0]	-	-	RES0, when N is not 0.

The following figure shows the DCADCRR bit assignments when reading the data cache tag RAM.

Figure 5-14: DCADCRR bit assignments when reading the data cache tag RAM



The following table shows the DCADCRR bit assignments when reading the data cache tag RAM.

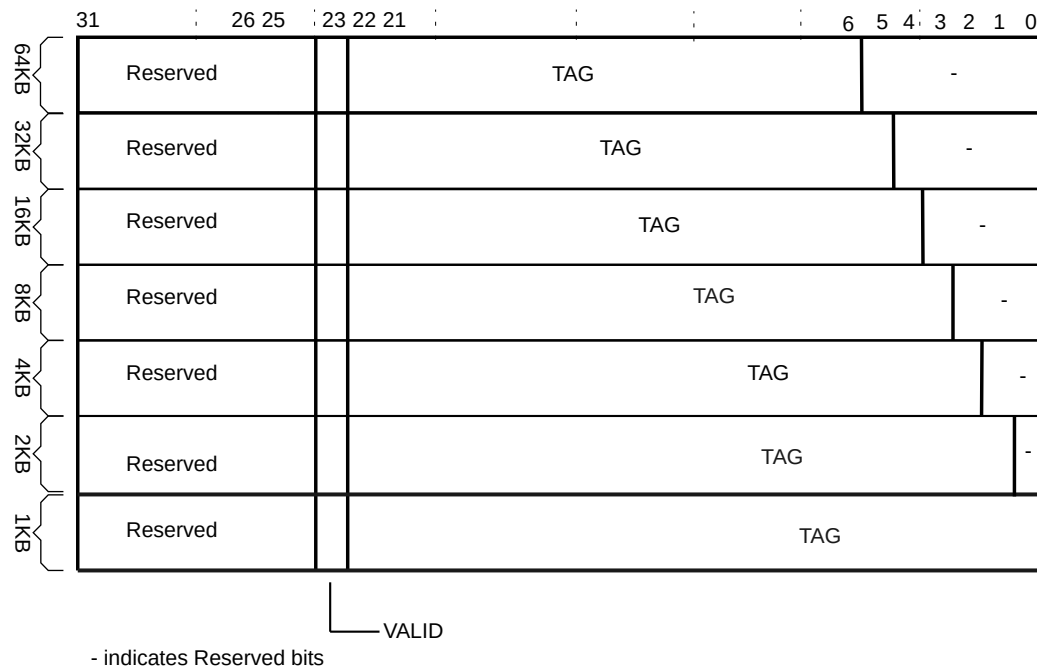
Table 5-20: DCADCRR bit assignments when reading the data cache tag RAM

Bits	Name	Type	Function
[31:27]	Reserved	-	RES0
[26]	NS-ATTR	RO	Non-secure attribute. Reserved when security extension is not present.

Bits	Name	Type	Function										
[25:23]	STATUS	RO	<p>Clean or dirty, transient, and outer attributes of the cache line. The attribute encoding is as follows:</p> <ul style="list-style-type: none"><li>0b000: Cache line is clean. Cache line is transient. Outer attributes of the cache line are <b>UNKNOWN</b>.</li><li>0b001: Cache line is clean. Cache line is not transient. Outer attributes of the cache line are <b>UNKNOWN</b>.</li><li>0b010: Cache line is dirty. Cache line is not transient. Outer attributes of the cache line are Non-cacheable.</li><li>0b011: Cache line is dirty. Cache line is not transient. Outer attributes of the cache line are Write-Back, Write Allocate.</li><li>0b100: Cache line is dirty. Cache line is not transient. Outer attributes of the cache line are Write-Back, No Write Allocate.</li><li>0b101: Cache line is dirty. Cache line is not transient. Outer attributes of the cache line are Write-Through, Write Allocate.</li><li>0b110: Cache line is dirty. Cache line is not transient. Outer attributes of the cache line are Write-Through, No Write Allocate.</li></ul> <p>0b111 is reserved.</p>										
[22]	VALID	RO	Valid state of the data cache line entry.										
[21:N]	TAG	RO	<p>Tag address. The number of significant bits of TAG depends on the data cache size.</p> <table><tr><td><b>64KB</b></td><td>N=4</td></tr><tr><td><b>32KB</b></td><td>N=3</td></tr><tr><td><b>16KB</b></td><td>N=2</td></tr><tr><td><b>8kB</b></td><td>N=1</td></tr><tr><td><b>4KB</b></td><td>N=0</td></tr></table>	<b>64KB</b>	N=4	<b>32KB</b>	N=3	<b>16KB</b>	N=2	<b>8kB</b>	N=1	<b>4KB</b>	N=0
<b>64KB</b>	N=4												
<b>32KB</b>	N=3												
<b>16KB</b>	N=2												
<b>8kB</b>	N=1												
<b>4KB</b>	N=0												
[N-1:0]	-	-	<b>RES0</b> , when N is not 0.										

The following figure shows the DCADCRR bit assignments when reading the unified cache tag RAM.

**Figure 5-15: DCADCRR bit assignments when reading the unified cache tag RAM**



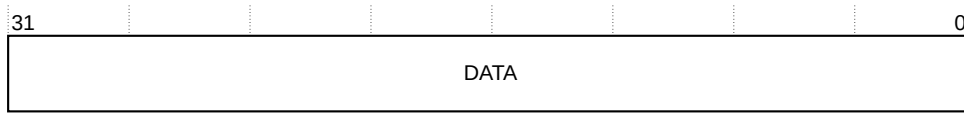
The following table shows the DCADCRR bit assignments when reading the unified cache tag RAM.

**Table 5-21: DCADCRR bit assignments when reading the unified cache tag RAM**

Bits	Name	Type	Function
[31:24]	-	-	RES0
[23]	VALID	RO	Valid state of the instruction cache line.
[22:N]	TAG	RO	Tag address. The number of significant bits of TAG depends on the unified cache size.  <div> <div>64KB</div> <div>32KB</div> <div>16KB</div> <div>8kB</div> <div>4KB</div> <div>2KB</div> <div>1KB</div> </div> <div> <div>N=6</div> <div>N=5</div> <div>N=4</div> <div>N=3</div> <div>N=2</div> <div>N=1</div> <div>N=0</div> </div>
[N-1:0]	-	-	RES0, when N is not 0.

The following figure shows the DCAICRR and DCADCRR bit assignments when reading the instruction cache, data cache, or unified cache data RAM.

**Figure 5-16: DCAICRR and DCADCRR bit assignments when reading the data RAM of instruction cache, data cache, or unified cache**



The following table shows the DCAICRR and DCADCRR bit assignments when reading the data RAM of instruction cache, data cache, or unified cache.

**Table 5-22: DCAICRR and DCADCRR bit assignments when reading the data RAM of instruction cache, data cache, or unified cache**

Bits	Name	Type	Function
[31:0]	DATA	RO	Data entry of instruction cache, data cache, or unified cache, ignoring <i>Error Correcting Code</i> (ECC).

## 5.13 Error bank registers

When the Cortex®-M52 processor is configured to support *Error Correcting Code* (ECC) logic, these registers record errors which occur during memory accesses to the L1 instruction cache, data cache, unified cache, and the TCM. They also allow certain memory locations to be locked so hard errors can be contained and corrected. Unified cache reuses the DEBRO and DEBR1.

The following table lists the error bank registers.

**Table 5-23: Error bank registers**

Address	Name	Type	Reset value	Description
0xE001E100	IEBRO	RW	0x00000000	IEBRO and IEBR1, Instruction Cache Error Bank Register 0-1
0xE001E104	IEBR1	RW	0x00000000	
0xE001E110	DEBRO	RW	0x00000000	DEBRO and DEBR1, Data Cache Error Bank Register 0-1
0xE001E114	DEBR1	RW	0x00000000	
0xE001E120	TEBRO	RW	0x00000000	TEBRO and TEBR1, TCM Error Bank Register 0-1
0xE001E124	TEBRDATA0	RO	0x00000000	Data for TCU Error Bank Register 0-1, TEBRDATA0 and TEBRDATA1
0xE001E128	TEBR1	RW	0x00000000	TEBRO and TEBR1, TCM Error Bank Register 0-1
0xE001E12C	TEBRDATA1	RO	0x00000000	Data for TCU Error Bank Register 0-1, TEBRDATA0 and TEBRDATA1

### 5.13.1 IEBRO and IEBR1, Instruction Cache Error Bank Register 0-1

The IEBRO and IEBR1 registers are the two error bank registers that are included for the L1 instruction cache. These registers are used to record errors that occur during memory accesses to the L1 instruction cache. They also allow certain memory locations to be locked so hard errors can be contained and corrected.

#### Usage Constraints

These registers are not banked between security states. If the Security Extension is implemented, this register is RAZ/WI from Non-secure state, and is only accessible from the Secure state.

These registers are only reset on Cold reset. Unprivileged access results in a BusFault exception.

#### Configurations

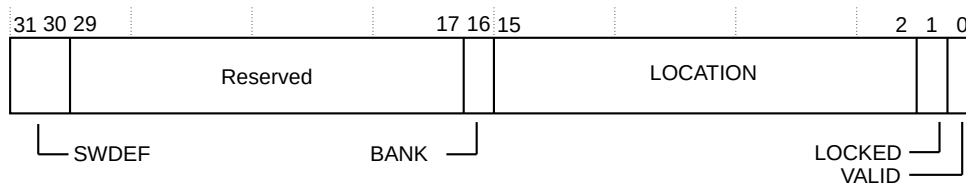
These registers are RAZ/WI if the L1 instruction cache is not present or if *Error Correcting Code* (ECC) is excluded.

#### Attributes

These registers are not banked between Security states. See [IMPLEMENTATION DEFINED registers summary](#) for more information.

The following figure shows the IEBRO and IEBR1 bit assignments.

**Figure 5-17: IEBRO and IEBR1 bit assignments**



The following table shows the IEBRO and IEBR1 bit assignments.

**Table 5-24: IEBRO and IEBR1 bit assignments**

Bits	Name	Type	Function
[31:30]	SWDEF	RW	User-defined register field. Error detection logic sets this field to 0b00 on a new allocation and on Cold reset.
[29:17]	Reserved	-	<b>RES0</b>
[16]	BANK	RW	Indicates which RAM bank to use.  0 Tag RAM. 1 Data RAM.
[15:2]	LOCATION	RW	Indicates the location in the L1 instruction cache RAM.  [15] Way [14:5] Index [4:2] Line word offset.

Bits	Name	Type	Function
[1]	LOCKED	RW	<p>Indicates whether the location is locked or not.</p> <p><b>0</b> Location is not locked and available for hardware to allocate.  <b>1</b> Software has locked the location and hardware is not allowed to allocate to this entry.</p> <p>Only one IEBrn register can be locked at any time. If one of these registers is already locked, then writing to the LOCKED bit of another is ignored. The Cold reset value is 0.</p>
[0]	VALID	RW	<p>Indicates whether the entry is valid or not.</p> <p><b>0</b> Entry is invalid.  <b>1</b> Entry is valid.</p> <p>The Cold reset value is 0.</p>

### 5.13.2 DEBR0 and DEBR1, Data Cache Error Bank Register 0-1

The DEBR0 and DEBR1 registers are the two error bank registers that are included for the L1 data cache and unified cache. These registers are used to record errors that occur during memory accesses to the L1 data cache and unified cache. They also allow certain memory locations to be locked so hard errors can be contained and corrected.

#### Usage Constraints

These registers are not banked between security states. If the Security Extension is implemented, this register is RAZ/WI from Non-secure state, and is only accessible from the Secure state.

These registers are only reset on Cold reset. Unprivileged access results in a BusFault exception.

#### Configurations

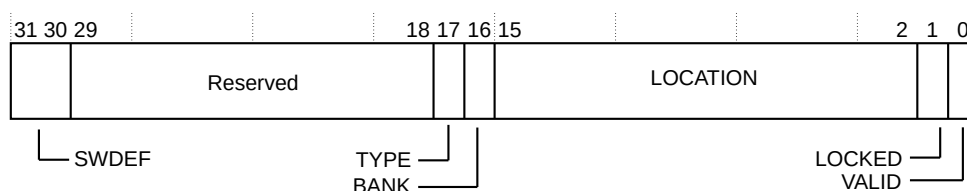
These registers are RAZ/WI if the L1 data cache and unified cache is not present or if *Error Correcting Code* (ECC) is excluded.

#### Attributes

These registers are not banked between Security states. See [IMPLEMENTATION DEFINED registers summary](#) for more information.

The following figure shows the DEBR0 and DEBR1 bit assignments.

**Figure 5-18: DEBR0 and DEBR1 bit assignments**



The following table shows the DEBR0 and DEBR1 bit assignments.

**Table 5-25: DEBR0 and DEBR1 bit assignments**

Bits	Name	Type	Function
[31:30]	SWDEF	RW	User-defined register field. Error detection logic sets this field to 0b00 on a new allocation and on Cold reset.
[29:18]	Reserved	-	<b>RES0</b>
[17]	TYPE	RW	<p>Indicates the error type.</p> <p><b>0</b> Single-bit error. <b>1</b> Multi-bit error.</p> <p>This field is only applicable to data cache.</p> <p>This field is <b>RES0</b> for unified cache configuration.</p>
[16]	BANK	-	<p>Indicates which RAM bank to use.</p> <p><b>0</b> Tag RAM. <b>1</b> Data RAM.</p>
[15:2]	LOCATION	-	<p>Indicates the location in the data cache or unified cache RAM. For data cache configuration:</p> <p><b>[15:14]</b> Way. <b>[13:5]</b> Index. <b>[4:2]</b> Line word offset.</p> <p>For unified cache configuration:</p> <p><b>[15]</b> Way. <b>[14:5]</b> Index. <b>[4:2]</b> Line word offset.</p>
[1]	LOCKED	RW	<p>Indicates whether the location is locked or not.</p> <p><b>0</b> Location is not locked and available for hardware to allocate. <b>1</b> Software has locked the location and hardware is not allowed to allocate to this entry.</p> <p>Only one DEBRn register can be locked at any time. If one of these registers is already locked, then writing to the LOCKED bit of another is ignored. The Cold reset value is 0.</p>
[0]	VALID	RW	<p>Indicates whether the entry is valid or not.</p> <p><b>0</b> Entry is invalid. <b>1</b> Entry is valid.</p> <p>The Cold reset value is 0.</p>

### 5.13.3 TEBRO and TEBR1, TCM Error Bank Register 0-1

The TEBRO and TEBR1 registers record the location of errors in the TCM.

#### Usage Constraints

These registers are not banked between security states. If the Security Extension is implemented, this register is RAZ/WI from Non-secure state, and is only accessible from the Secure state.

These registers are only reset on Cold reset. Unprivileged access results in a BusFault exception. These registers are RAZ/WI if *Error Correcting Code* (ECC) is excluded.

#### Configurations

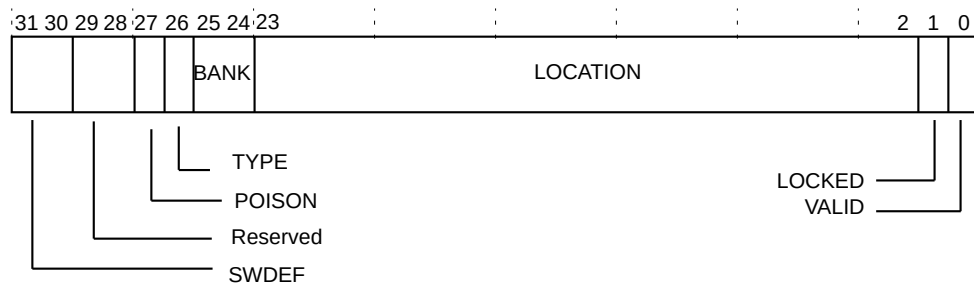
If *Error Correcting Code* (ECC) is excluded, these registers are RAZ/WI.

#### Attributes

These registers are not banked between Security states. See [IMPLEMENTATION DEFINED registers summary](#) for more information.

The following figure shows the TEBRO and TEBR1 bit assignments.

**Figure 5-19: TEBRO and TEBR1 bit assignments**



The following table shows the TEBRO and TEBR1 bit assignments.

**Table 5-26: TEBRO and TEBR1 bit assignments**

Bits	Name	Type	Function
[31:30]	SWDEF	RW	User-defined register field. Error detection logic sets this field to 0b00 on a new allocation and on Cold reset.
[29:28]	Reserved	-	<b>RES0</b>
[27]	POISON	RW	Indicates whether a BusFault is generated or not.  <b>0</b> Load or non-word store (RMW) to an address that hits this TEBR accesses the corresponding TEBRDATA register and does not get a BusFault. <b>1</b> Load to address that hits this TEBR gets a BusFault. Non-word store (RMW) to an address that hits this TEBR aborts the write.



Bits	Name	Type	Function
[26]	TYPE	RW	Indicates the error type.  <b>0</b> Single-bit error. <b>1</b> Multi-bit error.
[25:24]	BANK	RW	Indicates which RAM bank to use.  <b>0b00</b> DTCM0 <b>0b01</b> DTCM1 <b>0b10</b> ITCM  All other values are <b>RES0</b> .
[23:2]	LOCATION	RW	Indicates the physical location in the TCM bank.
[1]	LOCKED	RW	Indicates whether the location is locked or not.  <b>0</b> Location is not locked and available for hardware to allocate. <b>1</b> Software has locked the location and hardware is not allowed to allocate to this entry.  Only one TEBRn register can be locked at any time. If one of these registers is already locked, then writing to the LOCKED bit of another is ignored. The Cold reset value is 0.
[0]	VALID	RW	Indicates whether the entry is valid or not.  <b>0</b> Entry is invalid. <b>1</b> Entry is valid.  If software programs both TEBRn registers with the same LOCATION and BANK field values and VALID is set to 1, then the behavior of TCM accesses is <b>UNPREDICTABLE</b> . The Cold reset value is 0.

### 5.13.3.1 Data for TCU Error Bank Register 0-1, TEBRDATA0 and TEBRDATA1

The TEBRDATA0 and TEBRDATA1 registers provide storage for corrected data that is associated with an error.

#### Usage Constraints

If the Security Extension is implemented, this register is RAZ/WI from Non-secure state, and are only accessible from the Secure state.

These registers are only reset on Cold reset. Unprivileged access results in a BusFault exception.

If *Error Correcting Code* (ECC) is excluded, these registers are RAZ/WI.

#### Configurations

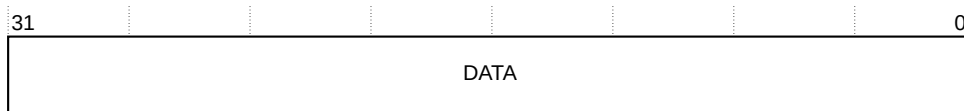
These registers are always implemented.

#### Attributes

These registers are not banked between Security states. See [IMPLEMENTATION DEFINED registers summary](#) for more information.

The following figure shows the TEBRDATA0 and TEBRDATA1 bit assignments.

**Figure 5-20: TEBRDATA0 and TEBRDATA1 bit assignments**



The following table shows the TEBRDATA0 and TEBRDATA1 bit assignments.

**Table 5-27: TEBRDATA0 and TEBRDATA1 bit assignments**

Bits	Name	Type	Function
[31:0]	DATA	RO	<p>The following access this register instead of the TCM location:</p> <ul style="list-style-type: none"> <li>• Loads and stores from software running on the processor, if the address matches the location in the corresponding TEBR.</li> <li>• Read and write transactions from the <i>AHB TCM Access</i> (TCM-AHB) interfaces, if the address matches the location in the corresponding TEBR.</li> </ul>

## 5.14 MSCR, Memory System Control Register

The MSCR controls the memory system features specific to the Cortex®-M52 processor.

### Usage constraints

If Security Extension is implemented and AIRCR.BFHFNMINS is zero, this register is RAZ/WI from the Non-secure state. Unprivileged access results in a BusFault exception.

### Configuration

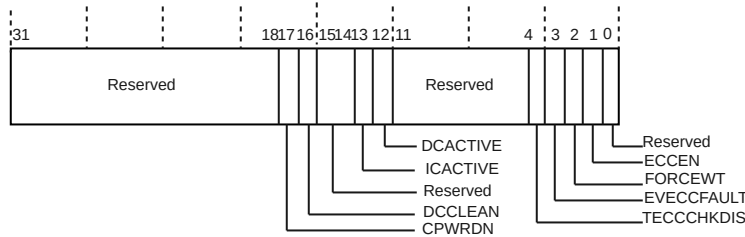
This register is always implemented.

### Attributes

This register is not banked between Security states. See [IMPLEMENTATION DEFINED registers summary](#) for more information.

The following figure shows the MSCR bit assignments.

**Figure 5-21: MSCR bit assignments**



The following table describes the MSCR bit assignments.

**Table 5-28: MSCR bit assignments**

Bits	Name	Type	Description
[31:18]	Reserved	-	<b>RES0</b>
[17]	CPWRDN	RO	<p>This bit indicates when the data and instruction caches (or Unified cache) are not accessible because they are either being powered down or being initialized using the automatic invalidation sequence. Software that is enabling the cache can use this bit to determine when the cache is available for use.</p> <p><b>0</b> Data and instruction cache (or Unified cache) in normal operational state.  <b>1</b> Data and instructions cache (or Unified cache) powered down or automatic invalidation sequence is in process.</p> <p>For on-line MBIST operations, Arm® recommends that PMC-100 is not programmed to carry out a memory test to the cache RAM when this field is 0, because the test will fail. If this occurs, a memory powered down error is indicated to the PMC-100.</p>
[16]	DCCLEAN	RW	<p>This bit indicates whether the data cache contains any dirty lines. The options are:</p> <p><b>0</b> L1 data cache contains at least one dirty line.  <b>1</b> L1 data cache does not contain any dirty lines.</p> <p>It is cleared to 0 on any write to the L1 data cache that sets the dirty bit.</p> <p>It is cleared to 1 at the end of any automatic L1 data cache invalidate all.  Software must only modify this register if it is restoring the state from before the core entered powerdown with the L1 data cache in retention.</p> <p>This field is not updated when a dirty line is evicted, therefore, MCSR.DCCLEAN can be 0, if the cache is currently clean but previously contained dirty data since the last time it was automatically invalidated.</p> <p>The reset value is 0.</p> <p>If the data cache is not included (no matter whether the unified cache is present or not), this field is RAZ/WI.</p>
[15:14]	Reserved	-	<b>RES0</b>

Bits	Name	Type	Description
[13]	ICACTIVE	RW	<p>This bit indicates whether the L1 instruction cache is active. The options are:</p> <p><b>0</b> L1 instruction cache is inactive. There is no allocation or lookups. Cache maintenance and direct cache access operations are treated as NOPs.</p> <p><b>1</b> L1 instruction cache is active. This implies normal behavior.</p> <p>The reset value is 1. If the L1 instruction cache is not included, this field is RAZ/WI.</p>
[12]	DCACTIVE	RW	<p>This bit indicates whether the L1 data cache is active. The options are:</p> <p><b>0</b> L1 data cache or unified cache is inactive. There is no allocation or lookups. Cache maintenance and direct cache access operations are treated as NOPs.</p> <p><b>1</b> L1 data cache or unified cache is active. This implies normal behavior.</p> <p>The reset value is 1. If neither L1 data cache nor unified cache is included, this field is RAZ/WI.</p>
[11:5]	Reserved	-	<b>RES0</b>
[4]	TECCCHKDIS	RW	<p>TCM ECC checking disable. This bit is intended to be used by SW to disable ECC checking, reporting, and correction during SW initialization of the TCM memories. This prevents false ECC errors being reported due to speculative TCM reads during SW initialization. Therefore, when this bit is b1, a TCM ECC error will not:</p> <ul style="list-style-type: none"> <li>• Cause a bus fault to be taken</li> <li>• Update the TEBR or RAS registers</li> <li>• Be reported on the DME bus</li> </ul> <p>0 = TCM ECC checking, reporting, and correction is enabled.</p> <p>1 = TCM ECC checking, reporting, and correction is disabled.</p> <p>Reset value is b0.</p> <p><b>Note:</b></p> <ul style="list-style-type: none"> <li>• This bit is RAZ/WI if ECC is not included in the processor configuration or if ECC is not enabled.</li> <li>• When SW TCM memory initialization is completed, SW must set this bit to b0.</li> <li>• ECC code generation for TCM writes is not affected by this bit.</li> </ul> <p>This bit does not affect on-line MBIST.</p>
[3]	EVECCFAULT	RW	<p>Enables asynchronous BusFault exceptions when data is lost on evictions. The options are:</p> <p><b>0</b> Asynchronous BusFaults are not generated when evicting lines with multi-bit errors in the data.</p> <p><b>1</b> Asynchronous aborts are generated when evicting lines with multi-errors in the data.</p> <p>This is intended for use in systems that do not support the AXI xPOISON signals. The reset value is 1. If ECC is not included, this field is RAZ/WI.</p>

Bits	Name	Type	Description
[2]	FORCEWT	RW	<p>Enables Forced Write-Through in the L1 data cache. The options are:</p> <p><b>0</b> Force Write-Through is disabled.  <b>1</b> Force Write-Through is enabled. All Cacheable memory regions are treated as Write-Through.</p> <p>The reset value is 0.            If the L1 data cache is not included, this field is RAZ/WI.</p>
[1]	ECCEN	RO	<p>Indicates whether <i>Error Correcting Code</i> (ECC) is present and enabled. The options are:</p> <p><b>0</b> ECC not present or not enabled.  <b>1</b> ECC present and enabled.</p> <p>The reset value depends on the <code>ECC</code> Verilog parameter and the external input signal <code>INITECCEN</code>. For more information on <code>ECC</code> Verilog parameter, see the <i>RTL configuration</i> section in the <i>Arm China Cortex®-M52 Processor Integration and Implementation Manual</i>.</p> <p>If ECC is not included, this field is RAZ/WI.</p>
[0]	Reserved	-	<b>RES0.</b>

## 5.15 PAHBCR, P-AHB Control Register

The PAHBCR enables accesses to *Peripheral AHB* (P-AHB) interface from software running on the processor. This register also provides information on the range of memory-mapped to the interface.

The P-AHB is always memory-mapped to a range of the Peripheral and Vendor\_SYS regions of the memory map. For more information on the memory map, see [Memory map](#).

## Usage Constraints

If the Security Extension is implemented and AIRCR.BFHFNMINS is zero, this register is RAZ/WI from Non-secure state. Unprivileged access results in a BusFault exception.

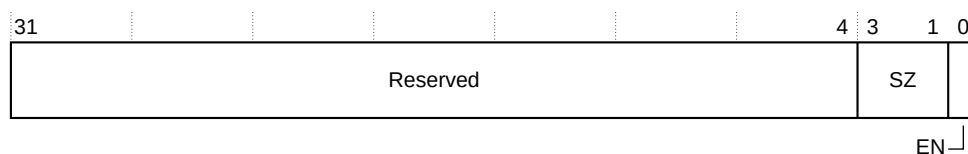
## Configuration

This register is always implemented.

## Attributes

See [IMPLEMENTATION DEFINED registers summary](#) for more information.

**Figure 5-22: PAHBCR bit assignments**



The following table shows the PAHBCR bit assignments.

**Table 5-29: PAHBCR bit assignments**

Bits	Name	Type	Description										
[31:4]	-	-	Reserved.										
[3:1]	SZ	RO	<p>P-AHB size. The options are:</p> <table><tr><td><b>0b000</b></td><td>0MB. This implies that P-AHB disabled.</td></tr><tr><td><b>0b001</b></td><td>64MB.</td></tr><tr><td><b>0b010</b></td><td>128MB.</td></tr><tr><td><b>0b011</b></td><td>256MB.</td></tr><tr><td><b>0b100</b></td><td>512MB.</td></tr></table> <p>Other encodings are reserved. At reset, the register field is loaded from the CFGPAHBSZ input signal. The CFGPAHBSZ signal determines the size of the peripheral port memory region.</p>	<b>0b000</b>	0MB. This implies that P-AHB disabled.	<b>0b001</b>	64MB.	<b>0b010</b>	128MB.	<b>0b011</b>	256MB.	<b>0b100</b>	512MB.
<b>0b000</b>	0MB. This implies that P-AHB disabled.												
<b>0b001</b>	64MB.												
<b>0b010</b>	128MB.												
<b>0b011</b>	256MB.												
<b>0b100</b>	512MB.												
[0]	EN	RW	<p>P-AHB enable. The options are:</p> <table><tr><td><b>0</b></td><td>P-AHB disabled. When disabled all accesses are made to the M-AXI interface.</td></tr><tr><td><b>1</b></td><td>P-AHB enabled.</td></tr></table> <p>The reset value is derived from the INITPAHBEN signal. This field only affects accesses in the Peripheral region of the memory map. Accesses from the Vendor_SYS region are always enabled.</p>	<b>0</b>	P-AHB disabled. When disabled all accesses are made to the M-AXI interface.	<b>1</b>	P-AHB enabled.						
<b>0</b>	P-AHB disabled. When disabled all accesses are made to the M-AXI interface.												
<b>1</b>	P-AHB enabled.												

## 5.16 Power mode control registers

The CPDLPSTATE and DPDLPSTATE registers allow software to control the required power mode of the functional and debug logic in the Cortex®-M52 processor.

The following table lists the power mode control registers.

**Table 5-30: Power mode control registers**

Address	Name	Type	Reset value	Description
0xE001E300	CPDLPSTATE	RW	0x00000303 <b>Note:</b> Bits [9:8] and [5:4] can be RAZ/WI depending on your processor implementation. See <a href="#">CPDLPSTATE, Core Power Domain Low Power State Register</a> for more information.	CPDLPSTATE, Core Power Domain Low Power State Register
0xE001E304	DPDLPSTATE	RW	0x00000003	DPDLPSTATE, Debug Power Domain Low Power State Register

### 5.16.1 CPDLPSTATE, Core Power Domain Low Power State Register

The CPDLPSTATE register specifies the required low-power states for core (PDCORE) and RAM (PDRAMS) power domains.

#### Usage Constraints

If AIRCR.BFHFNMINS is 0, then these registers are RAZ/WI from Non-secure state. Unprivileged access results in a BusFault exception.

#### Configurations

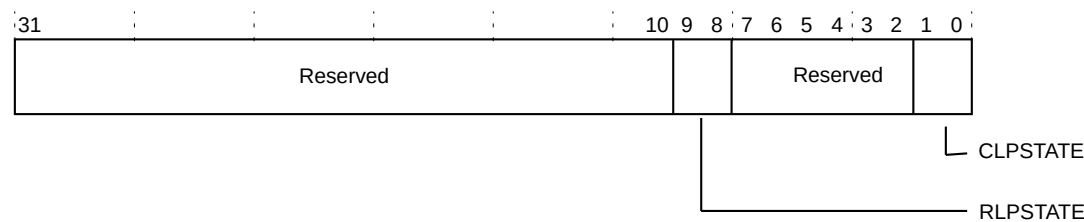
This register is always implemented.

#### Attributes

This register is not banked between Security states. See [IMPLEMENTATION DEFINED registers summary](#) for more information.

The following figure shows the CPDLPSTATE bit assignments.

Figure 5-23: CPDLPSTATE bit assignments



The following table shows the CPDLPSTATE bit assignments.

Table 5-31: CPDLPSTATE bit assignments

Bits	Name	Type	Function
[31:10]	Reserved	-	RES0

Bits	Name	Type	Function
[9:8]	RLPSTATE	RW	<p>Powerup state for PDRAMS power domain. This field indicates the minimum power mode that software requests. The actual requested power mode might depend on other conditions, for example, power domain activity. The actual transition of the power mode is performed by the P-Channel.</p> <p><b>0b00</b> ON.  <b>0b01</b> Reserved.  <b>0b10</b> Reserved.  <b>0b11</b> OFF.</p> <p><b>Note:</b>  This field is used only to control the Cache/No cache operating mode for the P-Channel. RAM retention is enabled by entering either of the following power modes:</p> <ul style="list-style-type: none"> <li>MEM_RET (Cache).</li> <li>FULL_RET (Cache).</li> </ul> <p>For more information, <a href="#">Core P-Channel and power mode selection</a>.</p> <p>If the L1 data cache and instruction cache are not present, this field is RAZ/WI.  The reset value is 0b11 on Cold reset.</p>
[7:2]	Reserved	-	<b>RES0</b>
[1:0]	CLPSTATE	RW	<p>Type of low-power state for PDCORE. This field indicates the minimum power mode that software requests. The actual requested power mode might depend on other conditions, for example, power domain activity. The actual transition of the power mode is performed by the P-Channel.</p> <p><b>0b00</b> ON. PDCORE is not in low-power state.  <b>0b01</b> ON, but the clock is off.  <b>0b10</b> RET.  <b>0b11</b> OFF.</p> <p>The reset value is 0b11 on Cold reset.</p>

## 5.16.2 DPDLSTATE, Debug Power Domain Low Power State Register

The DPDLSTATE register specifies the required low-power states for the debug (PDDEBUG) power domain.

### Usage Constraints

If the Security Extension is implemented and AIRCR.BFHFNMINS is 0, then these registers are RAZ/WI from Non-secure state.

### Configurations

This register is always implemented.

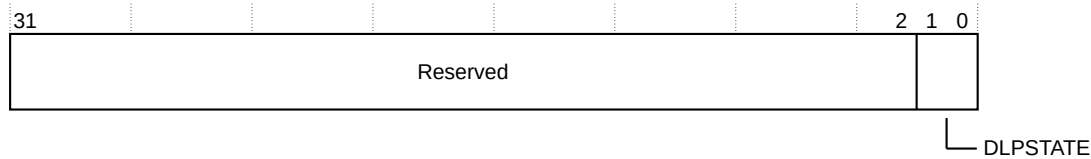
### Attributes

This register is not banked between Security states. See [IMPLEMENTATION DEFINED registers summary](#) for more information.

The following figure shows the DPDLSTATE bit assignments.



**Figure 5-24: DPDPSTATE bit assignments**



The following table shows the DPDPSTATE bit assignments.

**Table 5-32: DPDPSTATE bit assignments**

Bits	Name	Type	Function
[31:2]	Reserved	-	RES0
[1:0]	DLPSTATE	RW	<p>Type of low-power state for PDDEBUG. This field indicates the minimum power mode that software requests. The actual requested power mode might depend on other conditions, for example, power domain activity.</p> <p><b>0b00</b> ON. PDDEBUG is not in low-power state.  <b>0b01</b> ON, but the clock is off.  <b>0b10</b> RESERVED. Treated as ON, but clock OFF.  <b>0b11</b> OFF.</p> <p>The reset value is 0b11 at debug Cold reset, which is controlled by the nDBGRESET signal.</p>

## 5.17 Processor configuration information registers

The CFGINFOSEL and CFGINFORD registers provide information about the configuration of the processor including the values of all the Verilog parameters used during synthesis and input wire tie-off signals.

See [Cortex-M52 implementation options](#) for more information on the processor configuration options. For more detail on the RTL parameter values, see the *Cortex®-M52 Processor Integration and Implementation Manual*. The *Cortex®-M52 Processor Integration and Implementation Manual* is a confidential document that is available to licensees only and Arm® partners with an NDA agreement.

The following table lists the processor configuration information registers.

**Table 5-33: Processor configuration information registers**

Address	Name	Type	Reset value	Description
0xE001E700	CFGINFOSEL	WO	UNKNOWN	CFGINFOSEL, Processor configuration information selection register
0xE001E704	CFGINFORD	RO	UNKNOWN	CFGINFORD, Processor configuration information read data register

### 5.17.1 CFGINFOSEL, Processor configuration information selection register

The CFGINFOSEL register selects the configuration information which can then be read back using CFGINFORD.

#### Usage constraints

Unprivileged access results in a BusFault exception.

#### Configurations

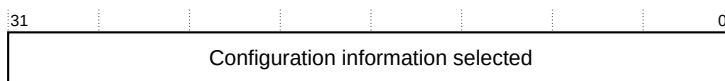
This register is always implemented.

#### Attributes

This register is banked between Security states. See [IMPLEMENTATION DEFINED registers summary](#) for more information.

The following figure shows the CFGINFOSEL bit assignments.

**Figure 5-25: CFGINFOSEL bit assignments**



The following table describes the CFGINFOSEL bit assignments.

**Table 5-34: CFGINFOSEL bit assignments**

Field	Name	Type	Description
[31:0]	Configuration information selected	WO	The value of this field depends on the configuration information selected.

The following table lists the CFGINFOSEL register value that depends on the configuration information selected. For more information on the configuration parameters that are listed in the following table, see [Cortex-M52 implementation options](#).

**Table 5-35: Configuration parameter selection used by the CFGINFOSEL register**

CFGINFOSEL value	Configuration information selected
0x1	ICACHESZ
0x2	DCACHESZ
0x3	ECC
0x4	FPMVE
0x5	Reserved
0x6	SECEXT
0x7	CPIF
0x8	MPU_NS
0x9	MPU_S

CFGINFOSEL value	Configuration information selected
0xA	SAU
0xB	ITGU
0xC	ITGUBLKSZ
0xD	ITGUMAXBLKS
0xE	DTGU
0xF	DTGUBLKSZ
0x10	DTGUMAXBLKS
0x11	NUMIRQ
0x12	IRQLVL
0x20+n, where $0 \leq n \leq 0xF$	Reserved
0x30+n, where $0 \leq n \leq 0xF$	IRQDIS[ (n*32)+31 : (n*32) ]
0x40	BUSPROT
0x41	LOCKSTEP
0x42	DBGLVL
0x43	ITM
0x44	ETM
0x45	PMC
0x46	PMCPROG SIZE
0x47	IWIC
0x48	WICLINES
0x49	CTI
0x4A	RAR
0x4B	INITL1RSTDIS
0x4C	CFGMEMALIAS
0x4D	CDECP
0x4E	CDERTLID
0x4F	PACBTI
0x50	FLOPPARITY
0x51	IDCACHEID
0x52	UCACHE
0x53	BUSAHB
0x54	CPUINST



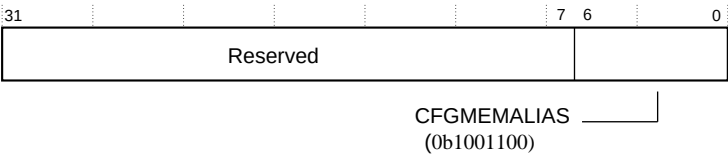
- INITL1RSTDIS and CFGMEMALIAS select the corresponding external input wire tie-off signal value.
- Input wire tie-off signals also affect the ECC, FPMVE, MPU\_NS, MPU\_S, and SAU values that are read. These signals are INITECCEN, CFGFPMVE, MPUNSDISABLE, MPUSDISABLE, and SAUDISABLE respectively. If the input wire tie-off disables the feature, then the configuration indicates that the feature is not supported.

- Parameters `IRQDIS` are selected across multiple values.
- `CDECP[7:0]` is mapped onto Verilog parameters `CDEMAPPEDONCP0-CDEMAPPEDONCP7` and input signal `CFGNOCDECP : CDECP[n] = CDEMAPPEDONCPn & ~CFGNOCDECP[n]`
- Size of unified cache should be read by setting `CFGINFOSEL` to `0x2`. Existence of unified cache should be read by setting `CFGINFOSEL` to `0x52`.

CFGINFOSEL register value examples

The following figure shows the `CFGINFOSEL` bit assignments when `CFGMEMALIAS` parameter is selected.

Figure 5-26: CFGINFOSEL bit assignments showing CFGMEMALIAS



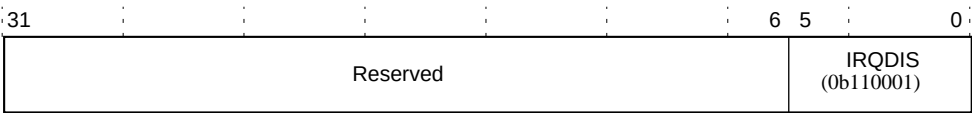
The following table describes the `CFGINFOSEL` bit assignments when `CFGMEMALIAS` parameter is selected.

Table 5-36: CFGINFOSEL bit assignments showing CFGMEMALIAS

Field	Name	Type	Description
[31:7]	Reserved	-	RES0
[6:0]	CFGMEMALIAS	WO	The value is 0x4C.

The following figure shows the `CFGINFOSEL` bit assignments when `IRQDIS[63:32]` parameter is selected and `n=1`.

Figure 5-27: CFGINFOSEL bit assignments showing IRQDIS when n=1



The following table describes the `CFGINFOSEL` bit assignments showing `IRQDIS[63:32]` when `n=1`.

**Table 5-37: CFGINFOSEL bit assignments showing IRQDIS when n=1**

Field	Name	Type	Description
[31:6]	Reserved	-	RES0
[5:0]	IRQDIS	WO	The value is 0x31, indicating IRQDIS [63:32] .

## 5.17.2 CFGINFORD, Processor configuration information read data register

The CFGINFORD register can be used to display the configuration information that the CFGINFOSEL register selects.

### Usage constraints

Unprivileged access results in a BusFault exception.

### Configurations

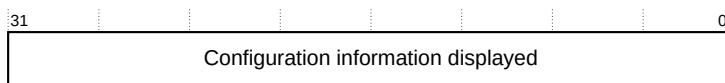
This register is always implemented.

### Attributes

This register is read-only and is banked between Security states. See [IMPLEMENTATION DEFINED registers summary](#) for more information.

The following figure shows the CFGINFORD bit assignments.

**Figure 5-28: CFGINFORD bit assignments**



The following table describes the CFGINFORD bit assignments.

**Table 5-38: CFGINFORD bit assignments**

Field	Name	Type	Description
[31:0]	Configuration information displayed	RO	The value of this field depends on the configuration information selected.

### CFGINFORD register value examples

The following figure shows the CFGINFORD bit assignments when the CFGINFOSEL register selects the CFGMEMALIAS parameter.

Figure 5-29: CFGINFORD bit assignments showing CFGMEMALIAS



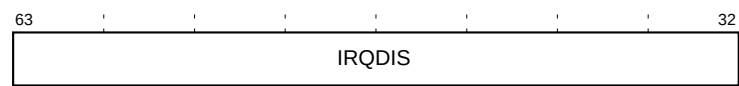
The following table describes the CFGINFORD bit assignments when CFGMEMALIAS configuration input signal is selected and the alias bit selected is 28.

Table 5-39: CFGINFORD bit assignments showing CFGMEMALIAS

Field	Name	Type	Description
[31:3]	Reserved	-	RES0
[2:0]	CFGMEMALIAS	RO	The value that is displayed is 0b100 to indicate that alias bit 28 has been selected.

The following figure shows the CFGINFORD bit assignments when IRQDIS parameter is selected and n=1.

Figure 5-30: CFGINFORD bit assignments showing IRQDIS when n=1



The following table describes the CFGINFOSEL bit assignments showing IRQDIS [63:32] when n=1. For this example, we are assuming that IRQDIS [63:32] is 0 for all interrupts, indicating lowest latency for IRQ32 to IRQ63.

Table 5-40: CFGINFORD bit assignments showing IRQDIS when n=1

Field	Name	Type	Description
[63:32]	IRQDIS	RO	0x00000000

## 5.18 ID\_PFR0, Processor Feature Register 0

The ID\_PFR0 register contains a field that indicates the version of the *Reliability, Availability, and Serviceability* (RAS) extension supported.

### Usage constraints

Unprivileged access results in a BusFault exception.

This register is accessible through unprivileged *Debug AHB* (D-AHB) debug requests when either DAUTHCTRL\_S.UIDAPEN or DAUTHCTRL\_NS.UIDAPEN is set.

### Configurations

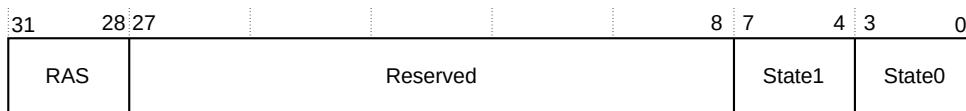
This register is always implemented.

### Attributes

This register is not banked between Security states. See [IMPLEMENTATION DEFINED registers summary](#) for more information.

The following figure shows the ID\_PFR0 bit assignments.

**Figure 5-31: ID\_PFR0 bit assignments**



The following table describes the ID\_PFR0 bit assignments.

**Table 5-41: ID\_PFR0 bit assignments**

Field	Name	Type	Description
[31:28]	RAS	RO	Identifies which version of the RAS architecture is implemented.  <b>0b0010</b> Version 1.
[27:8]	Reserved	-	<b>RES0</b>
[7:4]	State1	RO	T32 instruction set support.  <b>0b0011</b> T32 instruction set including Thumb-2 technology is implemented.
[3:0]	State0	RO	A32 instruction set support.  <b>0b0000</b> A32 instruction set is not implemented.

## 5.19 ITCMCR and DTCMCR, TCM Control Registers

The ITCMCR and DTCMCR registers enable access to the *Tightly Coupled Memories* (TCMs) by software running on the processor. These registers also provide information on the physical size of the memory connected.

### Usage Constraints

If the Security Extension is implemented and AIRCR.BFHFNMINS is 0, then these registers are RAZ/WI from Non-secure state. Unprivileged access results in a BusFault exception.

If the external input signal, LOCKTCM is asserted, these registers are read-only. For more information on LOCKTCM, see [Miscellaneous signals](#).

### Configuration

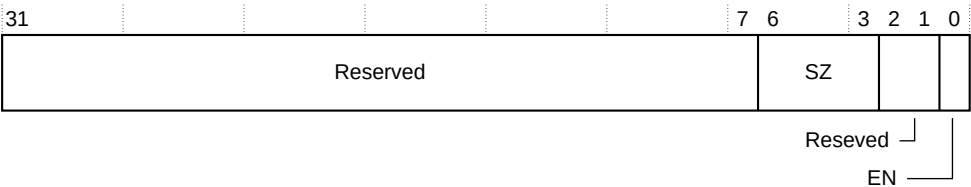
These registers are always implemented.

### Attributes

These registers are not banked between Security states. See [IMPLEMENTATION DEFINED registers summary](#) for more information.

The following figure shows the ITCMCR and DTCMCR bit assignments.

Figure 5-32: ITCMCR and DTCMCR bit assignments



The following table shows the ITCMCR and DTCMCR bit assignments.

Table 5-42: ITCMCR and DTCMCR bit assignments

Bits	Name	Type	Description
[31:7]	-	-	Reserved.



Bits	Name	Type	Description																												
[6:3]	SZ	RO	<p>TCM size indicates the size of the relevant TCM. The options are:</p> <table><tr><td>0b0000</td><td>No TCM implemented.</td></tr><tr><td>0b0011</td><td>4KB</td></tr><tr><td>0b0100</td><td>8KB</td></tr><tr><td>0b0101</td><td>16KB</td></tr><tr><td>0b0110</td><td>32KB</td></tr><tr><td>0b0111</td><td>64KB</td></tr><tr><td>0b1000</td><td>128KB</td></tr><tr><td>0b1001</td><td>256KB</td></tr><tr><td>0b1010</td><td>512KB</td></tr><tr><td>0b1011</td><td>1MB</td></tr><tr><td>0b1100</td><td>2MB</td></tr><tr><td>0b1101</td><td>4MB</td></tr><tr><td>0b1110</td><td>8MB</td></tr><tr><td>0b1111</td><td>16MB</td></tr></table> <p>All other encodings are reserved. The reset value is derived from the CFGITCMSZ and CFGDTCMSZ signals.</p>	0b0000	No TCM implemented.	0b0011	4KB	0b0100	8KB	0b0101	16KB	0b0110	32KB	0b0111	64KB	0b1000	128KB	0b1001	256KB	0b1010	512KB	0b1011	1MB	0b1100	2MB	0b1101	4MB	0b1110	8MB	0b1111	16MB
0b0000	No TCM implemented.																														
0b0011	4KB																														
0b0100	8KB																														
0b0101	16KB																														
0b0110	32KB																														
0b0111	64KB																														
0b1000	128KB																														
0b1001	256KB																														
0b1010	512KB																														
0b1011	1MB																														
0b1100	2MB																														
0b1101	4MB																														
0b1110	8MB																														
0b1111	16MB																														
[2:1]	Reserved	-	RAZ/WI.																												
[0]	EN	RW	<p>TCM enable. When a TCM is disabled, all accesses are made to the <i>AXI Main</i> (M-AXI) interface. The options are:</p> <table><tr><td>0</td><td>TCM disabled.</td></tr><tr><td>1</td><td>TCM enabled.</td></tr></table> <p>The reset value is derived from the INITTCMEN signal. This field only affects software accesses to the TCM. Accesses to the TCM from the TCM-AHB interface are always enabled.</p>	0	TCM disabled.	1	TCM enabled.																								
0	TCM disabled.																														
1	TCM enabled.																														

## 5.20 TCM security gate registers

The TCM security gates that are associated with the *Instruction Tightly Coupled Memory* (ITCM) and *Data Tightly Coupled Memory* (DTCM) are configured using the ITGU\_CTRL and DTGU\_CTRL registers, respectively. Additionally, there is a set of registers with a group of blocks, ITGU\_LUTn and DTGU\_LUTn. The configuration of a gate can be read from the read-only ITGU\_CFG and DTGU\_CFG registers.

The following table lists the TCM security gate registers.

**Table 5-43: TCM security gate registers**

Address	Name	Type	Reset value	Description
0xE001E500	ITGU_CTRL	RW	0x00000003	ITGU_CTRL and DTGU_CTRL, ITGU and DTGU Control Registers
0xE001E504	ITGU_CFG	RO	0xX0002X0X	ITGU_CFG and DTGU_CFG, ITGU and DTGU Configuration Registers

Address	Name	Type	Reset value	Description
0xE001E510+4n	ITGU_LUTn	<ul style="list-style-type: none"> <li>RW if <math>32n+1 &lt; 2^{\text{Number of ITGU blocks}}</math></li> <li>RO if <math>32n+1 \geq 2^{\text{Number of ITGU blocks}}</math></li> </ul>	0x00000000	ITGU_LUTn and DTGU_LUTn, ITGU and DTGU Look Up Table Registers
0xE001E600	DTGU_CTRL	RW	0x00000003	ITGU_CTRL and DTGU_CTRL, ITGU and DTGU Control Registers
0xE001E604	DTGU_CFG	RO	0xX0002X0X	ITGU_CFG and DTGU_CFG, ITGU and DTGU Configuration Registers
0xE001E610+4n	DTGU_LUTn	<ul style="list-style-type: none"> <li>RW if <math>32n+1 &lt; 2^{\text{Number of ITGU blocks}}</math></li> <li>RO if <math>32n+1 \geq 2^{\text{Number of ITGU blocks}}</math></li> </ul>	0x00000000	ITGU_LUTn and DTGU_LUTn, ITGU and DTGU Look Up Table Registers

### 5.20.1 ITGU\_CTRL and DTGU\_CTRL, ITGU and DTGU Control Registers

The ITGU\_CTRL and DTGU\_CTRL registers are the main *TCM Gate Unit* (TGU) control registers for the ITCM and DTCM respectively.

#### Usage constraints

If the Security Extension is implemented, these registers are RAZ/WI from the Non-secure state. Unprivileged access results in a BusFault exception. If the Security Extension is not implemented and TCM security gating is not included in the processor, then these registers are RAZ/WI.

If the external input signal LOCKITGU is asserted, the ITGU\_CTRL register is read-only.

If the external input signal LOCKDTGU is asserted, the DTGU\_CTRL register is read-only.

#### Configurations

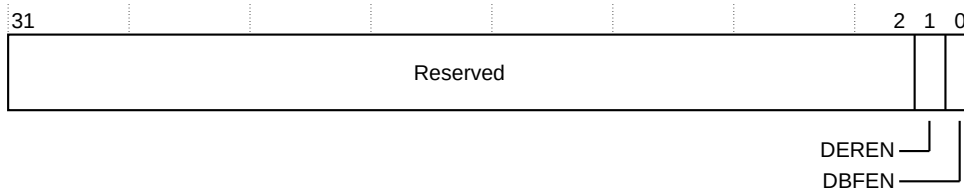
These registers are always implemented, but their behavior depends on whether the ITGU and DTGU are present.

#### Attributes

These registers are not banked between Security states. For more information, see [IMPLEMENTATION DEFINED registers summary](#).

The following figure shows the ITGU\_CTRL and DTGU\_CTRL bit assignments.

**Figure 5-33: ITGU\_CTRL and DTGU\_CTRL bit assignments**



The following table describes the ITGU\_CTRL and DTGU\_CTRL bit assignments.

**Table 5-44: ITGU\_CTRL and DTGU\_CTRL bit assignments**

Field	Name	Type	Description
[31:2]	Reserved	-	-
[1]	DEREN	RW	Enable <i>AHB TCM Access</i> (TCM-AHB) error response for TGU fault. The options are:  <b>0</b> Error response is not enabled. <b>1</b> Error response is enabled.
[0]	DBFEN	RW	Enable data side BusFault for TGU fault. The options are:  <b>0</b> BusFault not enabled. <b>1</b> BusFault enabled.

## 5.20.2 ITGU\_CFG and DTGU\_CFG, ITGU and DTGU Configuration Registers

The ITGU\_CFG and DTGU\_CFG registers allow the reading of configuration values for the ITGU and DTGU respectively.

### Usage constraints

If the Security Extension is implemented, these registers are RAZ/WI from the Non-secure state. Unprivileged access results in a BusFault exception. If the Security Extension is not implemented and TCM security gating is not included in the processor, then these registers are RAZ/WI.

### Configurations

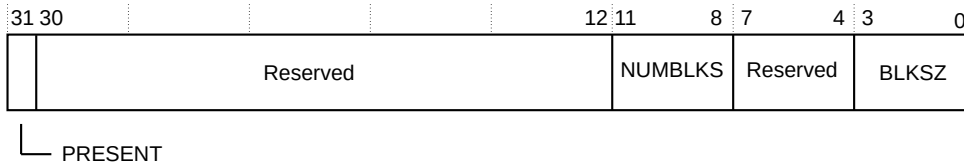
These registers are always implemented, but their behavior depends on whether the ITGU and DTGU are present.

### Attributes

These registers are not banked between Security states. For more information, see [IMPLEMENTATION DEFINED registers summary](#).

The following figure shows the ITGU\_CFG and DTGU\_CFG bit assignments.

**Figure 5-34: ITGU\_CFG and DTGU\_CFG bit assignments**



The following table describes the ITGU\_CFG and DTGU\_CFG bit assignments.

**Table 5-45: ITGU\_CFG and DTGU\_CFG bit assignments**

Field	Name	Type	Description
[31]	PRESENT	RO	This field determines if the TGU is present. The options are:  <b>0</b> TGU not present. <b>1</b> TGU is present
[30:12]	Reserved	-	<b>RES0</b>
[11:8]	NUMBLKS	RO	NUMBLKS=CFGxTCMSZ+4 -xTGU <sub>BLKSZ</sub> The number of TCM blocks is 2 <sup>NUMBLKS</sup> . Where: <ul style="list-style-type: none"> <li>CFGxTCMSZ is the configured TCM size.</li> <li>xTGU<sub>BLKSZ</sub> is the configured <i>Instruction Tightly Coupled Memory Gate Unit</i> (ITGU) or <i>Data Tightly Coupled Memory Gate Unit</i> (DTGU) block size.</li> </ul>
[7:4]	Reserved	-	<b>RES0</b>
[3:0]	BLKSZ	RO	TGU block size in bytes. This is 2 <sup>BLKSZ+5</sup> . This field is determined by the Verilog parameter xTGU <sub>BLKSZ</sub> .

### 5.20.3 ITGU\_LUTn and DTGU\_LUTn, ITGU and DTGU Look Up Table Registers

The ITGU\_LUTn and DTGU\_LUTn registers allows identifying the TGU blocks as Secure or Non-secure, where n is in the range 0-15.

#### Usage constraints

If the Security Extension is implemented, these registers are RAZ/WI from the Non-secure state. Unprivileged access results in a BusFault exception.

If the Security Extension is not implemented, then TCM security gating is not included in the processor and these registers are RAZ/WI.

If the external input signal LOCKITGU is asserted, the ITGU\_LUTn register is read-only.

If the external input signal LOCKDTGU is asserted, the DTGU\_LUTn register is read-only.

#### Configurations

The number of programmable blocks depends on the processor configuration and the physical TCM size. This is calculated using the following formula, where x is I for ITGU and D for DTGU:

$$N = 2^{x\text{ITGU\_CFG.NUMBLKS}}$$

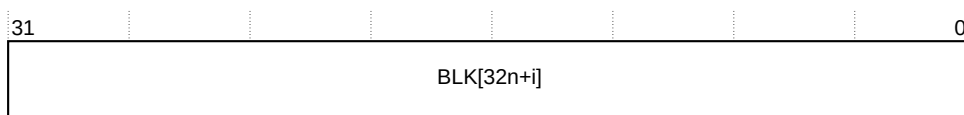
Accesses to register fields associated with blocks above the programmable number are treated as RAZ/WI. For more information on the ITGU\_CFG and DTGU\_CFG registers and the NUMBLKS field, see [ITGU\\_CFG and DTGU\\_CFG, ITGU and DTGU Configuration Registers](#).

### Attributes

These registers are not banked between Security states. For more information, see [IMPLEMENTATION DEFINED registers summary](#).

The following figure shows the ITGU\_LUTn and DTGU\_LUTn bit assignments.

**Figure 5-35: ITGU\_LUTn and DTGU\_LUTn bit assignments**



The following table describes the ITGU\_LUTn and DTGU\_LUTn bit assignments where:

- $0 \leq n \leq 15$
- $0 \leq i \leq 31$
- N is the number of programmable blocks:  $N = 2^{xTGU\_CFG.NUMBLKS}$
- x is I for ITGU and D for DTGU

**Table 5-46: ITGU\_LUTn and DTGU\_LUTn bit assignments for implemented block mapping**

Field	Name	Type	Description
[31:0]	BLK[32n+i]	RW for $32n+i < N$ RO for $32n+i \geq N$	<p>If <math>32n+i &lt; N</math>, then the block <math>32n+i</math> is implemented, and the security mapping bit options are:</p> <p><b>0</b> Block mapped as Secure <b>1</b> Block mapped as Non-secure</p> <p>If <math>32n+i \geq N</math>, then the block <math>32n+i</math> is not implemented, and the accesses are treated as RAZ/WI.</p>

### 5.20.3.1 ITGU\_LUTn and DTGU\_LUTn example

Consider the following example to calculate ITGU\_LUTn and DTGU\_LUTn, with ITGU\_CFG.NUMBLKS and DTGU\_CFG.NUMBLKS set to 4.

**Number of programmable blocks (N) =  $2^{xTGU\_CFG.NUMBLKS}$**

$xTGU\_CFG.NUMBLKS = CFGxTCMSZ + 4 - xTGUBLSZ$ , where x can be I or D for ITCM and DTCM respectively.

If  $CFGxTCMSZ$  is 0b011 and  $xTGUBLSZ$  is 3, then  $xTGU\_CFG.NUMBLKS$  is 4.

$N = 2^4$ , that is 16.

### Number of xTGU\_LUTn registers

Up to 16 xTGU\_LUTn registers can be configured which each register supporting 32 blocks, with n in the range 0-15. In this example, only one xTGU\_LUT register is required, that is, ITGU\_LUT and DTGU\_LUT, where n=0.

### Calculating the BLK[32n+i], where i is the bit offset in the register and can be in the range 0-31

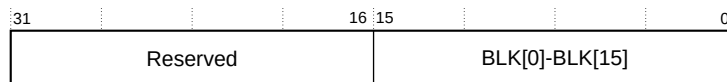
Since n=0 because all programmable blocks can fit into one 32-bit register, BLK is calculated as:

BLK[(32×0)+0] to BLK[(32×0)+15]. That is, BLK[0] to BLK[15].

### Bit assignments

The following figure shows the bit assignments for xTGU\_LUT when n=0.

**Figure 5-36: ITGU\_LUT and DTGU\_LUT bit assignments**



The following table describes the bit assignments.

**Table 5-47: ITGU\_LUTn and DTGU\_LUTn bit assignments for implemented block mapping**

Field	Name	Type	Description
[31:16]	-	RO	RAZ/WI.
[15:0]	BLK[0] to BLK[15]	RW	<p>If <math>32n+i &lt; N</math>, then the implemented block <math>32n+i</math> security mapping bit options are:</p> <p><b>0</b> Block mapped as Secure.</p> <p><b>1</b> Block mapped as Non-secure.</p>

## 5.21 EWIC interrupt status access registers

The *External Wakeup Interrupt Controller* (EWIC) interrupt status access registers, EVENTSPR, EVENTMASKA, and EVENTMASKn registers provide access to the *Nested Vectored Interrupt Controller* (NVIC) state that must be used to carry out software transfers to and from the EWIC in the system for sleep entry and exit when the automatic transfer feature is disabled.

The following table lists the EWIC interrupt status access registers.

**Table 5-48: EWIC interrupt status access registers**

Address	Name	Type	Reset value	Description
0xE001E400	EVENTSPR	WO	UNKNOWN	EVENTSPR, Event Set Pending Register

Address	Name	Type	Reset value	Description
0xE001E480	EVENTMASKA	RO	UNKNOWN	EVENTMASKA and EVENTMASKn, n=0-14, Wakeup Event Mask Registers
0xE001E484+4n	EVENTMASKn	RO	UNKNOWN	

### 5.21.1 EVENTSPR, Event Set Pending Register

The EVENTSPR is a write-only register that is used to set pending events at wakeup that cannot be directly set in the *Nested Vectored Interrupt Controller* (NVIC) using the architecture programming model.

#### Usage constraints

If the Security Extension is implemented and AIRCR.BFHFNMINS is zero, this register is RAZ/WI from the Non-secure state. Unprivileged access results in a BusFault exception.

#### Configurations

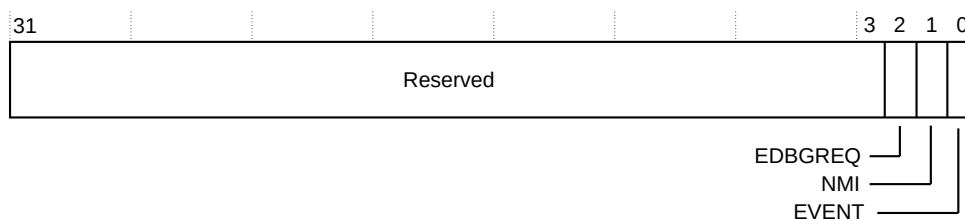
This register is always implemented.

#### Attributes

This register is not banked between Security states. For more information, see [IMPLEMENTATION DEFINED registers summary](#). The format of this register is identical to the EWIC\_PEND0 register. For more information on the EWIC\_PEND0 register, see [EWIC\\_PENDA and EWIC\\_PENDn, EWIC Pend Event Registers](#).

The following figure shows the EVENTSPR bit assignments.

**Figure 5-37: EVENTSPR bit assignments**



The following table describes the EVENTSPR bit assignments.

**Table 5-49: EVENTSPR bit assignments**

Field	Name	Type	Description
[31:3]	Reserved	-	RES0
[2]	EDBGREQ	WO	A write of one to this field causes the processor to behave as if an external debug request has occurred. A write of zero is ignored.
[1]	NMI	WO	A write of one to this field causes the processor to behave as if a non-maskable interrupt, NMI, has occurred. A write of zero is ignored.

Field	Name	Type	Description
[0]	EVENT	WO	A write of one to this field causes the processor to behave as if an RXEV event has occurred. A write of zero is ignored.

## 5.21.2 EVENTMASKA and EVENTMASKn, n=0-14, Wakeup Event Mask Registers

The EVENTMASKA and EVENTMASKn are read-only registers that provide the events on sleep entry which cause the processor to wake up. EVENTMASKA includes information about internal events and the EVENTMASKn registers cover external interrupt requests (IRQ). There is one register implemented for each of the 32 external interrupts that the *External Wakeup Interrupt Controller* (EWIC) supports. The EVENTMASKA register is always implemented.

### Usage constraints

If the Security Extension is implemented and AIRCR.BFHFNMINS is zero, this register is RAZ/WI from the Non-secure state. Unprivileged access results in a BusFault exception.

### Configurations

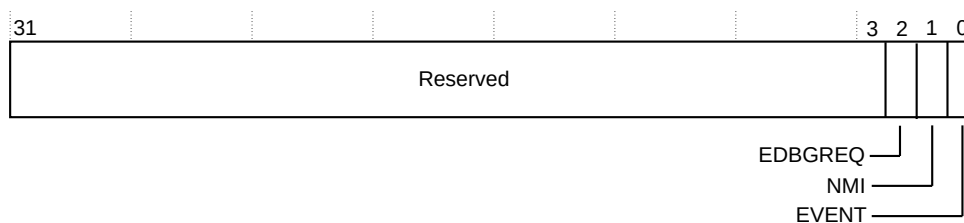
These registers are always implemented.

### Attributes

These registers are not banked between Security states. See [IMPLEMENTATION DEFINED registers summary](#) for more information.

The following figure shows the EVENTMASKA bit assignments.

**Figure 5-38: EVENTMASKA bit assignments**



The following table describes the EVENTMASKA bit assignments.

**Table 5-50: EVENTMASKA bit assignments**

Field	Name	Type	Description
[31:3]	-	-	Reserved, <b>RES0</b>
[2]	EDBGREQ	RO	Mask for external debug request. If this bit is 0, the mask is enabled.



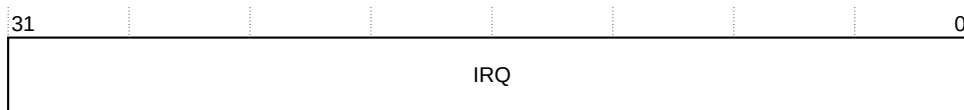
Field	Name	Type	Description
[1]	NMI	RO	Mask for NMI. If this bit is 0, the mask is enabled.  <b>Note:</b> An NMI can be masked in certain cases where the execution priority is equal to or higher than NMI priority.
[0]	-	-	<b>RES0</b>

EVENTMASKA[0] is **RES0** as the wakeup sensitivity to an external event is determined by the sleep entry instruction and not the processor state. The software transfer sequence must set the EWIC\_MASKA.EVENT register field, if the sleep entry instruction is WFE.

EWIC\_MASKA.EVENT should be set to 0b0 if the sleep entry instruction is not WFE. For more information on EWIC\_MASKA, see [EWIC\\_MASKA and EWIC\\_MASKn, EWIC Mask Registers](#).

[1The following figure shows the EVENTMASKn, where n=0-14, bit assignments.

**Figure 5-39: EVENTMASKn, where 0≤n<15, bit assignments**



The following table describes the EVENTMASKn, where n=0-14, bit assignments.

**Table 5-51: EVENTMASKn, where 0≤n<15, bit assignments**

Field	Name	Type	Description
[31:0]	IRQ	RO	Masks for interrupts (n×32) to ((n+1)×32)-1. If any of the bits are 0, the mask is enabled for the associated interrupt. Additionally, any interrupt that the WIC does not support is also RAZ.

## 5.22 STL observation registers

The Cortex®-M52 processor includes observation registers which can only be used by the *Software Test Library* (STL) to observe the internal state of the *Nested Vectored Interrupt Controller* (NVIC) priority tree outputs and to sample the *Memory Protection Unit* (MPU) region hit and associated attributes when a MemManage fault occurs on an instruction fetch or data access based on a programmable address.

For more information on STL, see the safety documentation associated to the processor IP. The safety documentation is part of the licensable processor IP safety package.

The following table lists the STL observation registers.

**Table 5-52: STL observation registers**

Address	Name	Type	Reset value	Description
0xE001E800	STLNVICPENDOR	RO	0x00000000	STLNVICPENDOR and STLNVICACTVOR, NVIC observation registers
0xE001E804	STLNVICACTVOR	RO	0x00000000	
0xE001E810	STLIDMPUSR	RW	0x00000000	STLIDMPUSR, STLIMPUOR, and STLDMPUOR, MPU observation registers
0xE001E814	STLIMPUOR	RO	0x00000000	
0xE001E818	STLDMPUOR	RO	0x00000000	

### 5.22.1 STLNVICPENDOR and STLNVICACTVOR, NVIC observation registers

The STLNVICPENDOR and STLNVICACTVOR registers can be used to observe the current output state of the NVIC pending and active priority tree which represents the highest priority pended or active interrupt at the point that the register is read.

#### Usage constraints

If the Security Extension is implemented, this register is RAZ/WI from the Non-secure state. Unprivileged access results in a BusFault exception.

#### Configurations

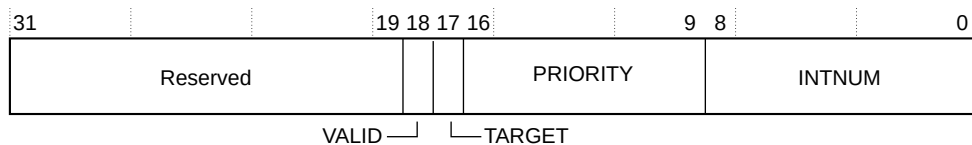
This register is always implemented.

#### Attributes

This register is not banked between Security states.

The following figure shows the STLNVICPENDOR and STLNVICACTVOR bit assignments.

**Figure 5-40: STLNVICPENDOR and STLNVICACTVOR bit assignments**



The following table describes the STLNVICPENDOR and STLNVICACTVOR bit assignments.

**Table 5-53: STLNVICPENDOR and STLNVICACTVOR bit assignments**

Field	Name	Type	Description
[31:19]	Reserved	-	RES0
[18]	VALID	RO	Priority tree output is valid.

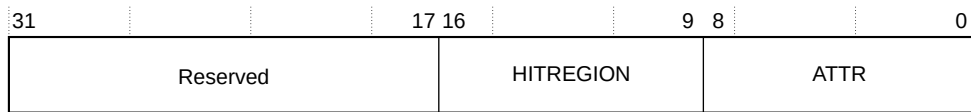


**Table 5-54: STLIDMPUSR bit assignments**

Field	Name	Type	Description
[31:5]	ADDR	RW	Sample address
[4:3]	Reserved	-	<b>RES0</b>
[2]	INSTR	RW	Select instruction MPU
[1]	DATA	RW	Select data channel 0 or data channel 1 MPU
[0]	Reserved	-	<b>RES0</b>

The following figure shows the STLIMPUOR and STLDMPUOR bit assignments.

**Figure 5-42: STLIMPUOR and STLDMPUOR bit assignments**



The following table describes the STLIMPUOR and STLDMPUOR bit assignments.

**Table 5-55: STLIMPUOR and STLDMPUOR bit assignments**

Field	Name	Type	Description
[31:17]	Reserved	-	<b>RES0</b>
[16:9]	HITREGION	RO	MPU region hit for data STLDMPUOR.  <b>Note:</b> <ul style="list-style-type: none"> <li>HITREGION range depends on the processor security state and MPU configuration</li> <li>HITREGION[7:4] is RAZ</li> </ul> This field is RAZ for STLIMPUOR.
[8:0]	ATTR	RO	Memory attributes  <div style="display: flex; justify-content: space-between;"> <div> <b>ATTR[8]</b>  <b>ATTR[7:4]</b>  <b>ATTR[3:0]</b> </div> <div> Shareability  Outer attributes  Inner attributes </div> </div> Inner and outer attributes use encoding from MAIR_ATTR, Memory Attribute Indirection Register Attributes in <i>Arm®v8-M Architecture Reference Manual</i> .



Note

- STLIMPUOR and STLDMPUOR are reset to 0x00000000 when the STLIDMPUSR register is updated
- STLIMPUOR and STLDMPUOR are updated independently if a fault is detected on the associated MPU if the associated selection fields in the STLIDMPUSR register is set. For example, if the sample register is configured to select the data

---

MPU, STLIDMPUSR.DATA is 0b1, then an access on channel D is captured in the appropriate observation register STLDMPUOR.

---

## 6. Initialization

This chapter describes how to initialize the Cortex®-M52 processor and which registers to access to enable functionality before using the processor features.

### 6.1 Initialization overview

Before you run your application, you might want to program values into registers and memory and enable certain processor features.

This chapter describes other initialization requirements, some of which are optional depending on the features you have implemented in the Cortex®-M52 processor.

### 6.2 Initializing and reprogramming the MPU

The Cortex®-M52 processor can be configured to include the *Memory Protection Unit* (MPU), which is an optional component that is primarily used for memory region protection.

If the Security Extension is included in the Cortex®-M52 processor, memory protection logic can be split between Secure and Non-secure MPU (MPU\_S and MPU\_NS).

Memory protection logic can be split between Secure and Non-secure MPU (MPU\_S and MPU\_NS).

The MPU\_CTRL.ENABLE must be set to 1 to enable the MPU.

If the Security Extension is included, then MPU\_CTRL\_NS is the Non-secure version of this register, and can be used to enable the Non-secure MPU region. For more information on MPU\_CTRL, see the *Arm®v8-M Architecture Reference Manual*.



For more information on the MPU, see [Memory Protection Unit](#).

---

#### Reprogramming the MPU

When setting up the MPU, and if it has been previously programmed, disable unused regions to prevent any old settings from affecting the latest MPU setup.

1. Execute a `DSB` instruction, to drain out any existing memory transactions.
2. Write to the MPU registers. For a complete list, see [Memory Protection Unit register summary](#).
3. Execute a `DSB` instruction and then an `ISB` instruction, to ensure that all subsequent memory accesses see the updated MPU setup.



Additionally, if any memory is converted from Cacheable to Non-cacheable or Device, and any write has been performed to that memory, you must perform data cache clean and invalidate operations (`DCIMVAC`) each of these cachelines.

For more information on these operations, see the *Arm®v8-M Architecture Reference Manual*.

## 6.3 Initializing the EPU

The *Extension Processing Unit* (EPU) is disabled on reset. The core must be in privileged mode to read from and write to the CPACR.

If the Security Extension is implemented, to allow the EPU to run Non-secure code, the NSACR must be setup by Secure privileged software.

The following code sequence demonstrates this:

```
NSACR EQU 0xE000ED8C
LDR R0, =NSACR ; Read NSACR
LDR r1, [R0] ; Set bits 10-11 to allow Non-secure access to CP10 and CP11
                coprocessors.
ORR R1, R1, #(0x3 << 10)
STR R1, [R0] ; Write back the modified value to the NSACR.
DSB
ISB ; Reset pipeline now the Non-secure access has been allowed to CP10 and CP11
                coprocessors.
```

To enable the EPU, privileged software must setup the CPACR, which is demonstrated by the following code sequence.



If the Security Extension is implemented, the CPACR is banked between Security states, this code sequence enables the EPU for the current Security state only.

```
CPACR EQU 0xE000ED88
LDR R0, =CPACR ; Read CPACR
LDR r1, [R0] ; Set bits 20-23 to enable CP10 and CP11 coprocessors
ORR R1, R1, #(0xF << 20)
STR R1, [R0] ; Write back the modified value to the CPACR
DSB
ISB ; Reset pipeline now the EPU is enabled.
```

## 6.4 Programming the SAU

If the Security Extension is included in the processor, the *Security Attribution Unit* (SAU) is available.

At reset, before any SAU regions are programmed, the default internal security level is selected using the SAU\_CTRL.ALLNS register. In the Cortex®-M52 processor, this register always resets to zero, setting most of the memory (except some regions in the PPB space) to Secure, and preventing an *Implementation Defined Attribution Unit* (IDAU) from overriding the security level.

However, after reset, Secure software can allow an IDAU to specify the security level for all memory regions by disabling all the SAU regions and setting SAU\_CTRL.ALLNS to one.

To enable the SAU, Secure software must:

1. Program the regions that are required into the SAU\_RBAR and SAU\_RLAR registers. To change an SAU region, you must clean and invalidate any addresses from the previous configuration from the cache.
2. Set the SAU\_CTRL.ENABLE bit to 1.

For more information on these registers, see *Arm®v8-M Architecture Reference Manual*.

The LOCKSAU signal prevents software accesses to the SAU registers. For more information on LOCKSAU, see [Miscellaneous signals](#).



For more information on the SAU and IDAU, see [Security Attribution Unit and Implementation Defined Attribution Unit](#)

---

## 6.5 Initializing the instruction and data cache

On initial powerup, the instruction and data caches are in an **UNKNOWN** state. Therefore, on initial powerup, the caches must be initialized either by automatic invalidation or through software invalidation.

If you implement RAM retention without using the P-Channel, then software invalidation of caches might be required.

If a P-Channel is not used for RAM retention, you must do either of the following:

- Set INITL1RSTDIS to an appropriate value when the cache is valid on reset
- Tie INITL1RSTDIS HIGH and invalidate software.

The caches are not accessible during the automatic invalidation sequence. Executing a DSB instruction causes the processor to wait for the sequence to complete.

The CCR.DC and CCR.IC register bits are banked based on security, therefore each Security state must set these bits to enable the data and instruction cache.

For more information on the CCR register, see *Arm®v8-M Architecture Reference Manual*.





Note

You can optionally implement *Error Correcting Code* (ECC) functionality on caches by setting the `ecc` RTL parameter. However, the Cortex®-M52 processor does not support disabling ECC using software. Enabling and disabling ECC is done at Cold reset by the `INITECCEN` signal. For more information on `INITECCEN`, see [Reset configuration signals](#).

For more information on instruction and data caches, see [Instruction cache, data cache, and unified cache](#).

## 6.5.1 Enabling the instruction and data cache

The following code sequence demonstrates how to enable the instruction and data cache for the current Security state when running in privileged mode. For unified cache, it reuses the CCR bit of data cache.

```
CCR EQU 0xE000ED14
LDR R0, =CCR ; Read CCR
LDR r1, [R0] ; Set bits 16 and 17 to enable D-cache and I-cache
ORR R1, R1, #(0x3 << 16)
STR R1, [R0] ; Write back the modified value to the CCR
DSB
ISB ; Perform DSB and ISB to guarantee change is visible to subsequent instructions
```

## 6.5.2 Powering down the caches

To powerdown the caches:

1. Set CCR.DC and CCR.IC to 0. CPDLPSTATE.RLPSTATE must be set to 0b11.
2. If the data cache contains dirty data that must be transferred to system memory, the entire cache must be cleaned with a set of Set/Way cache maintenance operations.

```
CCSIDR EQU 0xE000ED80 ; Current cache size ID register address
CSSELR EQU 0xE000ED84 ; Cache size selection register address
DCCSW EQU 0xE000EF6C ; Cache maintenance op address: data cache clean by set/way
; CSSELR selects the cache visible in CCSIDR
MOV r0, #0x0 ; 0 = select "level 1 data cache"
LDR r11, =CSSELR ;
STR r0, [r11] ;
DSB ; Ensure write to CSSELR before proceeding
LDR r11, =CCSIDR ; From CCSIDR
LDR r2, [r11] ; Read data cache size information
AND r1, r2, #0x7 ; r1 = cache line size
ADD r7, r1, #0x4 ; r7 = number of words in a cache line
UBFX r4, r2, #3, #10 ; r4 = number of "ways"-1 of data cache
UBFX r2, r2, #13, #15 ; r2 = number of "set"-1 of data cache
CLZ r6, r4 ; calculate bit offset for "way" in DCISW
LDR r11, =DCCSW ; clean cache by set/way
inv_loop1 ; For each "set"
MOV r1, r4 ; r1 = number of "ways"-1
LSLS r8, r2, r7 ; shift "set" value to bit 5 of r8
inv_loop2 ; For each "way"
LSLS r3, r1, r6 ; shift "way" value to bit 30 in r6
ORRS r3, r3, r8 ; merge "way" and "set" value for DCISW
STR r3, [r11] ; invalidate D-cache line
```

```
SUBS r1, r1, #0x1 ; decrement "way"
BGE inv_loop2 ; End for each "way"
SUBS r2, r2, #0x1 ; Decrement "set"
BGE inv_loop1 ; End for each "set"
DSB ; Data sync barrier after invalidate cache
ISB ; Instruction sync barrier after invalidate cache
```

3. Set MSCR.DCACTIVE and MSCR.IACTIVE to 0. As a result, the processor core deasserts bit 16 of the COREPACTIVE signal, which is a hint to the external power controller that PDRAMS can be powered down.

### 6.5.3 Powering up the caches

To powerup the caches:

1. Set MSCR.DCACTIVE and MSCR.IACTIVE to 1. As a result, the processor core asserts COREPACTIVE[16], to indicate to an external power controller that PDRAMS are required to be powered up.
2. Set CCR.DC and CCR.IC to 1. After the external power control logic has powered up PDRAMS, the *Core Power Control* (CPC) triggers an automatic invalidation of the RAMs (if INITL1RSTDIS is 0), and after that is complete, subsequent instructions can cause allocations to and lookups in the caches.

## 6.6 Enabling the branch cache

The branch cache is disabled on reset. You must enable the branch cache to implement *Low Overhead Branch* (LOB) Extension.

The processor core must be in privileged mode to read from and write to the CCR. If the Security Extension is implemented, the CCR.LOB bit is banked so it must be enabled for each Security state that uses the LOB Extension. For more information on CCR, see the *Arm®v8-M Architecture Reference Manual*.

The following code sequence demonstrates how to enable the branch cache for the current Security state when running in privileged mode.

```
CCR EQU 0xE000ED14
LDR R0, =CCR ; Read CCR
LDR r1, [R0] ; Set bits 19 to enable LOB
ORR R1, R1, #(0x8 << 16)
STR R1, [R0] ; Write back the modified value to the CCR
DSB
ISB ; Reset pipeline now LOB is enabled.
```

## 6.7 Enabling and preloading the TCM

The Cortex®-M52 processor can optionally include *Tightly Coupled Memories* (TCMs).

### Enabling the TCMs

For more information, see [TCM interfaces](#).

Software must set the ITCMCR.EN and DTCMCR.EN fields to enable access to the *Instruction Tightly Coupled Memory* (ITCM) and *Data Tightly Coupled Memory* (DTCM) respectively. For more information on these registers, see [ITCMCR and DTCMCR, TCM Control Registers](#).

Alternatively, if the INITTCMEN[1:0] signal is asserted on Cold or Warm reset, then software does not need to write to these registers. For more information on the INITTCMEN[1:0] signal, see [Reset configuration signals](#).

### Preloading the TCMs

The methods to preload the TCMs are:

#### Memory copy with running boot code

When boot code includes a memory copy routine that reads data from a ROM and writes it into the appropriate TCM, you must enable the TCM to perform this operation. This bootcode must be run from an address outside the TCM region.

#### DMA into TCM

You can use a *Direct Memory Access* (DMA) device that reads data from a ROM and writes it to the TCMs through the *AHB TCM Access* (TCM-AHB) interface. This method can be used to preload the TCM so they can be used by the processor from reset.

#### Using the TCM from reset

If the TCM interface is configured to enable the TCMs at reset and the reset vector address is inside the TCM memory region, then the processor boots from TCM. The system must ensure that the bootcode software is present in the appropriate memory region before execution starts. This can be accomplished by either initializing the memory before reset or by transferring the data after reset using the TCM-AHB interface and asserting the CPUWAIT input signal. Asserting this signal stops the processor fetching or executing instructions after reset. When the CPUWAIT signal is deasserted the processor starts fetching instructions from the reset vector address in the normal way.



Note

Asserting CPUWAIT only takes effect when the processor is under processor reset or Cold reset, that is, nSYSRESET or nPORESET is asserted. The processor does not halt if CPUWAIT is asserted while the processor is running.

The ITCM and DTCM can be locked from software access using the external input signal, LOCKTCM. When this signal is asserted, it disables writes to registers that are associated with the TCM region from software or from a debug agent connected to the processor.

- ITCMCR.

- DTCMCR.

Asserting this signal prevents changes to the TCM configuration. All writes to the registers are ignored.



When ECC is enabled, before performing a byte, halfword, or unaligned word write to a TCM location which causes an RMW, you must initialize the location first by performing an aligned word or doubleword write to the location. Arm® recommends that all TCM locations are initialized in this manner by boot code.

---

## 6.8 Enabling and locking the TCM security gates

TCM gating is enabled by tying the external input signal CFGMEMALIAS to a non-zero value.

The *TCM Gate Unit* (TGU) can be locked from software access using the external input signals LOCKITGU and LOCKDTGU. When these signals are asserted the corresponding TGU registers become read-only. This allows a TGU configuration to be programmed and then locked from further changes by software. For more information on TCM security gating, see [TCM Gate Units](#).

## 6.9 Enabling the P-AHB interface

Software can enable the *Peripheral AHB* (P-AHB) interface by writing to the PAHBCR.EN register.

For more information on PAHBCR, see [PAHBCR, P-AHB Control Register](#).

Alternatively, you can assert INITPAHBEN HIGH at Cold or Warm reset, to enable the P-AHB interface. If you do this, there is no need for a software write to PAHBCR.EN. For more information on INITPAHBEN, see [Reset configuration signals](#).

The P-AHB can be locked from software access using the external input signal, LOCKPAHB. When this signal is asserted, writes to PAHBCR register from software or from a debug agent connected to the processor are disabled and the register becomes read-only. Asserting this signal prevents changes to P-AHB port enable status in PAHBCR.EN.

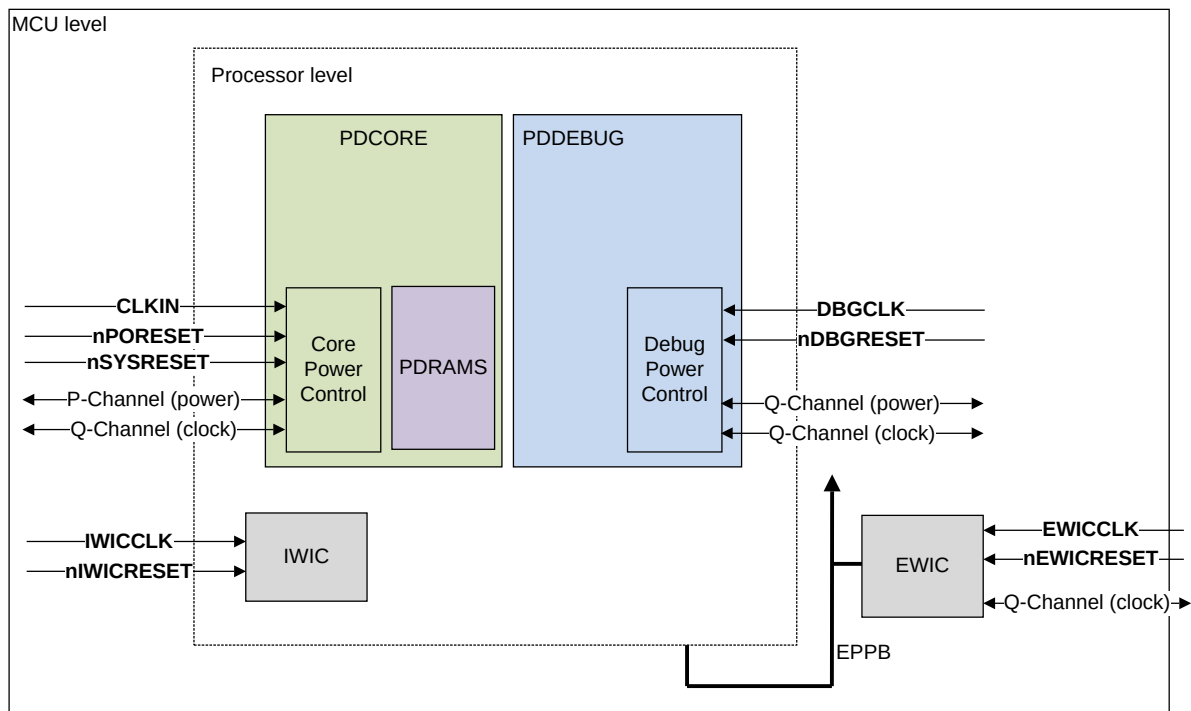
## 7. Power management

This chapter introduces Cortex®-M52 processor power management concepts.

### 7.1 Power domains

The Cortex®-M52 processor can be partitioned into power domains as shown in the following figure.

**Figure 7-1: Cortex®-M52 processor power domains**



The power domains are described in the following table.

**Table 7-1: Power domain description**

Power domain	Description
PDCORE	This contains the processor core, L1 memory system, the <i>Cross Trigger Interface</i> (CTI), <i>Nested Vectored Interrupt Controller</i> (NVIC), and <i>Extension Processing Unit</i> (EPU) logic, that is, the floating-point and M-profile Vector Extension (MVE) logic.
PDRAMS	This contains the L1 instruction cache and data cache RAMs.
PDDEBUG	This contains most of the debug logic. It includes the <i>BreakPoint Unit</i> (BPU), <i>Data Watchpoint and Trace</i> (DWT), <i>Instrumentation Trace Macrocell</i> (ITM), and <i>Embedded Trace Macrocell</i> (ETM).

- The *Internal Wakeup Controller* (IWIC) is located in a separate power domain, the IWIC power domain, that might be on when the processor core is powered down, to allow the detection of wakeup events.
- The MCU level in the processor deliverable includes an example *External Wakeup Controller* (EWIC). The EWIC can be placed in any point in the system that is considered to be Always-on relative to the processor domain.
- The IP deliverable that is shipped does not support any power domains at the MCU level, and the MCU level is considered to be relatively always-on to the processor domain. You can use the delivered MCU and customize your system to include appropriate power domains depending on your implementation.
- If the processor is configured to include DCLS functionality, then the redundant core is part of the PDCORE domain

## 7.2 Power states

The power domains in the Cortex®-M52 processor can be in ON, OFF, or RET power states. The RET power state requires the processor logic to be implemented with state retention.

The following table shows the supported power states.

**Table 7-2: Supported power states**

Power state	Clocks running	Processor logic powered	Register and RAM contents retained	Reset asserted
ON	Yes/No	Yes	Yes	No
RET	No	No	Yes	No
OFF	No	No	No	Yes

The following table shows the permitted Cortex®-M52 processor power states for the power domains.

**Table 7-3: Permitted power states for Cortex®-M52 processor power domains**

Power state	PDCORE	PDDEBUG	PDRAMS
ON	Permitted	Permitted	Permitted
RET	Permitted	Not permitted	Permitted. <b>Note:</b> Retention in the PDRAMS domain is only supported when the processor is in the MEM_RET (Cache) or FULL_RET (Cache) power modes.
OFF	Permitted	Permitted	Permitted

Not all power state combinations are permitted. The combination of PDCORE and PDRAMS power states is called the power mode. PDDEBUG is independent of the other power domains. It can either be ON or OFF, regardless of the processor power mode.



When a power domain is in the ON power state, if the clock is not running, then the domain is considered to be in low-power state.

## 7.3 Power and operating mode transitions

The Cortex®-M52 processor power modes are based on the Arm standard modes and encodings. The power modes are extended with operating modes, which control whether the L1 instruction and data caches in the PDRAMS domain are enabled.

The Arm® standard modes and encodings are defined in the *Arm® Power Control System Architecture* specification. The *Arm® Power Control System Architecture* specification is a confidential document that is only available to licensees and Arm® partners with an NDA agreement.

An external power controller controls the processor power and operating mode through the P-Channel. An external clock controller controls the Q-Channel allowing system-level clock gating. The P-Channel and the clock control Q-Channel are connected to the *Core Power Control* (CPC) in the PDCORE domain. The CPC manages the internal clocking and reset of the PDCORE and PDRAMS domains. It supports the clock and reset signals that are described in [Clock and clock enable signals](#) and [Reset signals](#), and system-level clock gating. The processor indicates the minimum required power mode according to its state and internal control registers using the COREPACTIVE signal. For more information on COREPACTIVE, see [P-Channel and Q-Channel power control signals](#) and [COREPACTIVE signal encoding](#).

An external power controller controls the debug power mode through the debug domain power control Q-Channel. The debug domain power control Q-Channel is connected to the *Debug Power Control* (DPC) in the PDDEBUG domain. A clock control Q-Channel is also available to support high-level clock gating.

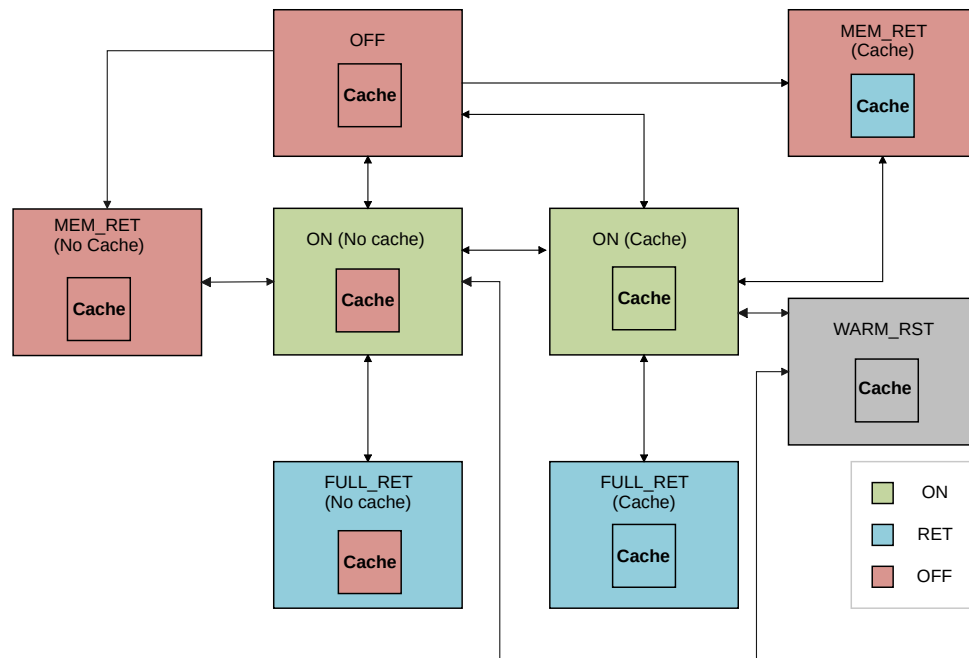
Only certain transitions between power and operating modes are allowed. [Power and operating mode transitions](#) shows these permitted transitions. If an external power controller request is made to move between two modes which are not directly connected, then the request is denied (using COREPDENY).

Retention in the PDRAMS domain depends on the overall power and operating mode. RAM retention is selected by entering any of the following:

- MEM\_RET (Cache).
- FULL\_RET (Cache).

In other power modes, the PDRAMS state depends on the operating mode.

**Figure 7-2: Permitted power and operating modes and transitions**



When the COREPACTIVE signal indicates a required move between two modes which are not directly connected, the external power controller must transition through one or more intermediate modes to get to the final required power and operating mode. When only a change in PDRAMs is required even if the change involves moving through multiple power and operating modes, the processor supports this and indicates the required intermediate transitions using the COREPACTIVE signals. See [Operating mode transitions which change PDRAMs power state](#).

The following table describes the power and operating modes that are shown in [Permitted power and operating modes and transitions](#).

**Table 7-4: Power and operating mode transitions**

Power and operating mode	Description
ON (Cache)	Full Run mode with cache powered on.
ON (No cache)	Full Run mode and cache powered off.
FULL_RET (Cache)	All functional logic and cache in retention. This mode is software transparent powerdown.
FULL_RET (No cache)	All functional logic in retention with cache powered off (if present), or the cache is not present. This mode is software transparent powerdown.



Power and operating mode	Description
MEM_RET (Cache)	All functional logic is powered off, RAMs in retention.
MEM_RET (No cache)	MEM_RET (No cache) is functionally identical to OFF. The power mode and associated transitions are included for compatibility with the Arm® CoreLink™ PCK-600 Power Control Kit <i>Power Policy Unit</i> (PPU). The Cortex®-M52 processor never requests this state using the P-Channel COREACTIVE output signal.
OFF	Powered off, shutdown mode.
WARM_RST	Warm reset.

In [Power and operating mode transitions](#), the No cache operating mode implies that if your system configuration includes caches, then the cache is present, but disabled and powered OFF. The following register bits are set to appropriate values:

- MSCR.ICAACTIVE and MSCR.DCAACTIVE are 0.
- CPDLPSTATE.RLPSTATE is 0b11.



Note

- A transition from OFF to MEM\_RET is allowed. Arm® recommends this as being required for full compatibility with the Arm® CoreLink™ PCK-600 Power Control Kit *Power Policy Unit* (PPU). Transitions from MEM\_RET to OFF are not allowed. The system is responsible for maintaining power in the RAMs to ensure that processor cache content is preserved when entering MEM\_RET.
- A transition from OFF to MEM\_RET or MEM\_RET to ON does not invalidate the cache even when INITL1RSTDIS is set to 0.
- MEM\_RET (No cache) is functionally identical to OFF. The state and associated transitions are included for compatibility with current Arm® CoreLink™ PCK-600 Power Control Kit *Power Policy Unit* (PPU). The Cortex®-M52 processor never requests this state using the P-Channel COREACTIVE output signal.
- A request on the P-Channel to transition to the current power mode is always accepted.

### 7.3.1 Operating mode transitions which change PDRAMS power state

The processor supports transitions between operating modes where the PDRAMS domain is enabled or disabled.

For example, if the operating mode is ON (No cache) the processor can request to enable PDRAMS. This request results in COREACTIVE[16] being asserted, requesting a transition to ON (Cache), but the other bits on COREACTIVE remain static. The transition between ON (Cache) and ON (No cache) is called a change of operating mode.

The CoreLink™ PCK-600 Power Control Kit *Power Policy Unit* (PPU) supports dynamic transitions between operating modes only when in the ON power mode. The *Core Power Control* (CPC) logic includes a secondary state-machine which transitions COREACTIVE through the ON power mode to allow the external power controller to enable or disable PDRAMS.

## 7.4 Core P-Channel and power mode selection

The power modes are based on the power state of the PDCORE and PDRAMS domains.

The requested power mode is defined according to the lowest achievable mode based on the processor logic state, external conditions, and the corresponding CPDLPSTATE register fields. The resulting power mode is driven on the P-Channel COREPACTIVE output signal.



For more information on COREPACTIVE signal encoding, see [COREPACTIVE signal encoding](#).

The following table shows the resultant overall power mode that is based on the requests from each individual processor power domain.

**Table 7-5: Requested domain power states and resultant power and operating mode**

Requested domain power states		Resultant power and operating mode
PDCORE	PDRAMS	
ON	ON	ON (Cache)
ON	OFF	ON (No cache)
ON	RET	ON (Cache)
RET	ON	FULL_RET (Cache)
RET	OFF	FULL_RET (No cache)
RET	RET	FULL_RET (Cache)
OFF	ON	MEM_RET (Cache)
OFF	OFF	MEM_RET (No cache)
OFF	RET	MEM_RET (Cache)
OFF	OFF	OFF
-	-	WARM_RST

Some combinations of power domain states do not map directly onto a power mode. The lowest possible power mode is selected which matches the requested PDRAMS power state.

At Cold reset, the internal power mode is OFF and the P-Channel COREPACTIVE signal is also driven OFF. Before fetching the reset vector or starting to execute instructions, the processor waits for the system to request or initialize an operational state for the PDCORE domain.

The following power modes are supported on the P-Channel for device state initialization at reset deassertion:

- OFF.
- MEM\_RET.
- ON.

A period  $t_{init}$  is defined in device clock cycles after which the device is guaranteed to have sampled the P-Channel COREPSTATE input signal for all possible valid reset states. For the Cortex®-M52 processor,  $t_{init}$  is three cycles of CLKIN.

### 7.4.1 P-Channel interface tie-off when P-Channel is not used

When the P-Channel is not used in the system, there are some tie-off requirements that must be met.

The following table shows the P-Channel interface tie-off when P-Channel interface is not used.

**Table 7-6: P-Channel interface tie-off when P-Channel interface is not used**

P-Channel signal	Tie-off values when P-Channel interface is not used
COREPSTATE	<p>The value can be any of the following:</p> <ul style="list-style-type: none"> <li>0b11000, indicating the power and operating mode is ON (Cache)</li> <li>0b01000, indicating the power and operating mode is ON (No cache)</li> </ul>
COREPREQ	0b0

If the P-Channel is not used in the system and the interface input signals are tied-off, the processor transitions to ON power mode out of Cold reset and starts executing instructions.



COREPSTATE must be configured only to ON (No Cache) if the instruction and data caches have not been configured in the processor.

The parameters ICACHESZ[4:0] and DCACHESZ[4:0] must be set to 0b00000.

Otherwise, processor behavior is **UNPREDICTABLE**.

## 7.5 COREPACTIVE and required power mode

The *Core Power Control* (CPC) unit in the PDCORE power domain determines the required minimum power mode and drives this mode on the P-Channel COREPACTIVE output signal.



For more information on the COREPACTIVE output signal encoding, see [COREPACTIVE signal encoding](#).

The required power mode is a combination of the processor state and the CPDLPSTATE register. This combination allows software to select the required low-power state for each power domain.

The CPDLPSTATE register controls the three types of low-power state. The low-power states are:

- OFF
- RET
- ON with the clock off



If present, external coprocessors are included in the requirements for moving the PDCORE domain to low-power state.

The CPDLPSTATE register can be used to select low power states based only on stopping the clock input to the PDCORE domain, CLKIN. The Q-Channel that is associated with CLKIN drives the CLKINQACTIVE signal LOW providing a hint to the system that the CLKIN Q-Channel might accept a quiescence request, therefore, allowing the clock to be gated if:

- All the low-power requirements for the PDCORE domain are true apart from the value of CPDLPSTATE.
- The CPDLPSTATE field CLPSTATE is not 0b00.

The individual required power states are translated to one of the overall power modes that are given in [Requested domain power states and resultant power and operating mode](#) and used to drive the COREPACTIVE signal. The following table describes the COREPACTIVE and COREPSTATE bits encoding.

**Table 7-7: COREPSTATE and COREPACTIVE bits encodings**

Processor power mode	Standard power mode	COREPSTATE[4] (With cache)	COREPSTATE[3:0]	COREPACTIVE[16] (With cache)	COREPACTIVE[8:0] most significant set bit
WARM_RST	WARM_RST	-	0b1001	0	-
ON (Cache)	ON	1	0b1000	1	8
ON (No cache)	ON	0	0b1000	0	8
FULL_RET (Cache)	FULL_RET	1	0b0101	1	5
FULL_RET (No cache)	FULL_RET	0	0b0101	0	5
MEM_RET (Cache)	MEM_RET	1	0b0010	1	2
MEM_RET (No cache)	MEM_RET	0	0b0010	0	2
OFF	OFF	-	0b0000	COREPACTIVE is driven to 0.	



- COREPACTIVE[16] and COREPSTATE[4] indicates the cache state. If the cache operating mode is requested, COREPACTIVE[16] or COREPSTATE[4] is HIGH.
- COREPACTIVE bits 0, 1, 3, 9-15, and 17-20 are not used. They are always tied LOW.

- COREPSTATE values not listed in [COREPSTATE and COREPACTIVE bits encodings](#) are invalid. If a system attempts to transition to one of these encodings, the P-Channel responds with COREPDENY.
- For more information on WARM\_RST, see [Warm reset power mode](#).
- Power modes WARM\_RST and OFF are independent from COREPSTATE[4]. The processor behaves identically whether this bit is 1 or 0.
- The processor uses a different name for the MEM\_OFF encoding in the Arm® because the corresponding power mode affects the EPU rather than memory, but maintains compatibility with the PPU power mode.

## 7.5.1 COREPACTIVE signal encoding

The following table shows the COREPACTIVE signal encoding.

**Table 7-8: COREPACTIVE signal encoding**

Signal bit	Encoding
[20:17]	Unused
[16]	Indicates requirement for cache ON state
[15:9]	Unused
[8]	ON
[5]	FULL_RET
[3]	Unused
[2]	MEM_RET
[1]	Unused
[0]	Unused

## 7.6 PDCORE low-power requirements

The following conditions must be true to request a PDCORE low-power state on the COREPACTIVE signal using the P-Channel:

- The processor is in sleep mode.
- SCR.SLEEPDEEP is set.
- WICCONTROL[0] is asserted so that SLEEPDEEP means *Wakeup Interrupt Controller* (WIC) sleep.
- If *External Wakeup Interrupt Controller* (EWIC) is configured, any automatic WIC loading must be completed.
- The *AHB TCM Access* (TCM-AHB) interface is inactive.
- The *Debug AHB* (D-AHB) interface is inactive.

- The processor core is not halted.
- CPDLPSTATE.CLPSTATE is not equal to 0b00.
- No MBIST operation is in progress.
- The CTI is not included or disabled, or if the CTI is included and enabled, there is no valid mapping that is set up for an external cross trigger to be passed onto the processor, or CTI integration mode is not enabled in CTI\_ITCONTROL.

When the PDCORE low-power requirements are met, CPDLPSTATE.CLPSTATE selects the low-power state.

- 
- If the Security Extension is included in the processor:
    - The input signal, CPSPRESENT[n] indicates that coprocessor n is included
    - CPACR\_S.CPn and CPACR\_NS.CPn indicate that coprocessor n is enabled and needs power.

If the Security Extension is not included in the processor:

- The input signal, CPNSPRESENT[n] indicates that coprocessor n is included
- CPACR\_NS.CPn indicates that coprocessor n is enabled and needs power.



Note

For more information on CPACR, see the *Arm®v8-M Architecture Reference Manual*.

- If a coprocessor CPn that is included in the system is indicating that the state cannot be lost (CPSPRESENT[n]&&CPPWR.SUn=0b0), then a request to powerdown in CPDLPSTATE.CLPSTATE is converted to RET to preserve the coprocessor state. For more information on CPPWR, see the *Arm®v8-M Architecture Reference Manual*.
  - To request a PDCORE low-power state using clock gating only, CPDLPSTATE.CLPSTATE must be 0b01.
- 

## 7.7 PDRAMS powerdown requirements

The following conditions must be true to powerdown PDRAMS:

- MSCR.DCACTIVE is equal to 0b0. This field is ignored for transparent retention of the RAMs.
- MSCR.ICACTIVE is equal to 0b0. This field is ignored for transparent retention of the RAMs.
- CPDLPSTATE.RLPSTATE is equal to 0b11.
- No cache maintenance operation is in progress.
- Automatic cache invalidation is not active.
- No MBIST operation is in progress to the instruction cache or data cache.

The low-power state is selected using CPDLPSTATE.RLPSTATE.

## 7.8 Warm reset power mode

The WARM\_RST power mode is used when external control logic requires the processor to be put in a safe state for Warm reset.

Asserting Warm reset (nSYSRESET) is allowed when PDCORE is in power mode OFF or MEM\_RET. Applying the reset in any other mode (except for WARM\_RST mode) will have an **UNPREDICTABLE** effect.

Asserting nSYSRESET when PDCORE is in an active state and not in WARM\_RST state might result in system deadlock.

### Entering WARM\_RST

WARM\_RST can only be entered when the PDCORE domain is powered on, corresponding to the ON power mode. Requesting WARM\_RST from any other power mode results in COREPDENY being asserted.

The processor asserts COREPACCEPT when PDCORE is transitioning to a quiescent state, and is held asserted until core quiescence is achieved. Therefore, it is only safe to assert nSYSRESET after the P-Channel transition to WARM\_RST is completed.

This core quiescence requires that there are no outstanding transactions on the AXI Main (M-AXI)/AHB Main (M-AHB), *Peripheral AHB* (P-AHB), *External Private Peripheral Bus* (EPPB), *Debug AHB* (D-AHB), and *AHB TCM Access* (TCM-AHB) interfaces. If a request is made on the TCM-AHB interface while the processor is in WARM\_RST power mode, it is ignored. Therefore, the system is responsible for ensuring that no accesses are made on the TCM-AHB interface until the processor leaves WARM\_RST whether or not reset is asserted in the power mode.

If a debug access is made on D-AHB while in WARM\_RST it is captured on the slave interface and pended until the power mode is switched back out of WARM\_RST, at which point the access is made to the processor. If the D-AHB access is to state which has been reset while in WARM\_RST then the result could be **UNPREDICTABLE**.

The processor ensures that all the outputs of the PDCORE domain are set to their reset values. Therefore, when nSYSRESET is asserted these values do not change, which helps to prevent reset domain crossing issues.

In particular, the AIRCR.SYSRESETREQ is cleared on entry to WARM\_RST, so that the SYSRESETREQ output signal is driven to 0 matching the reset condition.

Warm reset can always be applied safely when the processor is in a low-power sleep state with all power domains powered-on and no requests are active on the TCM-AHB or D-AHB interfaces.

If your system has a P-Channel interface for power control, then it is only safe to assert nSYSRESET when the processor is in any of the following modes:

- WARM\_RST, which is advantageous because it does not require software support

- OFF
- MEM\_RET

If your system does not use a P-Channel interface for power control, then Arm® recommends that you assert nSYSRESET when the processor core is in sleep mode, all the power domains are powered up, and there are no TCM-AHB or D-AHB requests.

The Warm reset request does not require that any of the power domains change power state. The combination of power states remains unchanged from when the processor entered the WARM\_RST power mode.

### Exiting WARM\_RST

The processor can exit WARM\_RST mode, whether or not nSYSRESET has been asserted to reset the PDCORE power domain. If no reset has occurred program execution continues from where it was before WARM\_RST was requested.

The processor asserts COREPACCEPT for any request to transition from WARM\_RST to the ON power modes. Requests to transition from WARM\_RST to any other power mode results in COREPDENY being asserted.

The WARM\_RST request does not require that any of the power domains change power state. The combination of power states when in the WARM\_RST power mode will be the same as before it entered that power mode. The COREPACTIVE output signal remains the same value as it was before COREPACCEPT was asserted for COREPSTATE indicating WARM\_RST entry.



Note

The Cortex®-M52 processor has internal logic that deals with any metastability caused by either of the following asynchronous resets:

- Asserting nSYSRESET while the processor core is in the WARM\_RST, OFF, or MEM\_RET power modes.
- Resetting any power domain because of entry to a power state that is controlled by the P-Channel or Q-Channel.

## 7.9 Debug Q-Channel and PDDEBUG power domain

A Q-Channel interface controls the PDDEBUG power domain.

The PDDEBUG power domain logic drives the PWRDBGQACTIVE signal HIGH to indicate that the domain is active if any of the following conditions are met:

- Trace is enabled, DEMCR.TRCENA=1.
- If configured, the *Embedded Trace Macrocell* (ETM) is enabled, TRCPDCR.PU=1.
- There is outstanding trace data in the ETM, *Instrumentation Trace Macrocell* (ITM), or *Data Watchpoint and Trace* (DWT).



- There is an outstanding access to any of the registers in PDDEBUG from software or from a debug request on *Debug AHB* (D-AHB).
- The *BreakPoint Unit* (BPU) is enabled, FP\_CTRL.ENABLE=1.
- DPDLPSTATE.DLPSTATE is 0b00 or 0b01.



Note

- Setting DPDLPSTATE.DLPSTATE to 0b01 indicates that DBGCLK can be gated when the domain is idle. This results in the DBGCLKQACTIVE signal being set LOW when the PDDEBUG domain is idle.
- For more information on TRCPDCR, see *Arm China CoreSight™ ETM-M52 Technical Reference Manual*.
- For more information on the FP\_CTRL and DEMCR, see the *Arm®v8-M Architecture Reference Manual*.
- For more information on the Q-Channel interface and its signals, see the *AMBA® Low Power Interface Specification*.

## 7.10 Q-Channel clock control

To optimize power usage, the Cortex®-M52 processor includes Q-Channel interfaces which allow the system to gate the clocks that are associated with the PDCORE and PDDEBUG power domains at a high level in the clock tree.

The PDCORE clock signal, CLKIN, is controlled using:

- CLKINQREQn.
- CLKINQACCEPTn.
- CLKINQDENY.
- CLKINQACTIVE.

The PDDEBUG clock signal, DBGCLK, is controlled using:

- DBGCLKQREQn.
- DBGCLKQACCEPTn.
- DBGCLKQDENY.
- DBGCLKQACTIVE.

The following rules apply for PDCORE and PDDEBUG clock signals:

- If both CLKIN and DBGCLK are running, they must be fully synchronous to each other.
- CLKINQACTIVE is asserted when PDCORE requires a clock.
- DBGCLKQACTIVE is asserted when PDDEBUG requires a clock.
- CLKIN can only be gated when its clock control Q-Channel is in the *Q\_STOPPED* state or when the PDCORE P-Channel is in *FULL\_RET*, *MEM\_RET*, or *OFF*.

- DBGCLK can only be gated when its clock control Q-Channel is in *Q\_STOPPED* state or when the PDDEBUG power control Q-Channel is in *Q\_STOPPED*.

## 7.11 PWRDBGWAKEQACTIVE

The PDCORE domain asserts the PWRDBGWAKEQACTIVE output signal for the following cases.

- When there is an access to a register located in the PDDEBUG domain, either from software running on the processor or from a request on the *Debug AHB* (D-AHB) interface.
- When there is a request to any *External Private Peripheral Bus* (EPPB) address which is not a part of the *External Wakeup Interrupt Controller* (EWIC) address space starting from 0x0047000.

This signal must be routed to the external power controller and used to power up the PDDEBUG domain. The processor uses an internal signal to determine when the debug domain is active and when it is safe to perform the access. The PWRDBGWAKEQACTIVE signal can be OR gated with the PWRDBGQACTIVE signal to indicate to the external power controller that the PDDEBUG domain must be activated.

## 8. Memory model

This chapter describes the Cortex®-M52 processor memory model.

### 8.1 Memory map

The default memory map for the Cortex®-M52 processor covers the range 0x00000000-0xFFFFFFFF.

**Table 8-1: Default memory map**

Address Range (inclusive)	Region	Interface
0x00000000-0x1FFFFFFF	Code	All accesses are performed on the <i>Instruction Tightly Coupled Memory</i> (ITCM) or Main interface.
0x20000000-0x3FFFFFFF	SRAM	All accesses are performed on the <i>Data Tightly Coupled Memory</i> (DTCM) or Main interface.
0x40000000-0x5FFFFFFF	Peripheral	<ul style="list-style-type: none"> <li>Data accesses are performed on <i>Peripheral AHB</i> (P-AHB) or Main interface.</li> <li>Instruction accesses are performed on Main interface.</li> </ul>
0x60000000-0x9FFFFFFF	External RAM	All accesses are performed on the Main interface.
0xA0000000-0xDFFFFFFF	External device	All accesses are performed on the Main interface.
0xE0000000-0xE00FFFFF	PPB	<ul style="list-style-type: none"> <li>Instruction fetches are not supported.</li> <li>Reserved for system control and debug.</li> <li>Data accesses are either performed internally or on <i>External Private Peripheral Bus</i> (EPPB).</li> </ul>
0xE0100000-0xFFFFFFFF	Vendor_SYS	<ul style="list-style-type: none"> <li>Instruction fetches are not supported.</li> <li>0xE0100000-0xEFFFFFFF is reserved.</li> <li>Vendor resources start at 0xF0000000.</li> <li>Data accesses are performed on P-AHB interface.</li> </ul>

#### Security states for memory requests

The AMBA® interfaces on the Cortex®-M52 processor include support for indicating the security level of a memory request for the following interfaces:

**Table 8-2: Security signals used in Cortex®-M52 memory interfaces**

Interface	AMBA® standard	Security signals
M-AXI	AMBA 5 AXI	ARPROT[1], AWPROT[1].
M-AHB	AMBA 5 AHB5	HNONSECC, HNONSECSYS
P-AHB	AMBA 5 AHB5	HNONSECP
EPPB	AMBA 4 APB	PPROT[1]

When the Security Extension is included, the security attribute of a memory request depends on the Security state of the processor and the regions defined in the internal *Secure Attribution Unit*

(SAU) or an external *Implementation Defined Attribution Unit* (IDAU). However, in some areas of the memory map, the security level of data accesses are determined only by the Security state.

If the Security Extension is not included, all memory is treated as Non-secure.

See the *Arm®v8-M Architecture Reference Manual* for more information about the memory model.

The TCM interfaces do not include signals indicating the security level of a transaction. Instead, the processor includes an internal security gate to support programmable regions, which conform to the *Trusted Base System Architecture* (TBSA) for Armv8-M. This security gating mechanism is described in [TCM and P-AHB security access control](#).

## Bit-banding

This feature is not supported on the Cortex®-M52 processor unless your system includes additional hardware to perform the appropriate mapping. If bit-banding support is required, Arm® recommends that peripherals are memory mapped to alias their bits to byte, halfword, or words.

## 8.2 Memory types

Each address in the memory map has a memory type which is determined by the default memory map or the *Memory Protection Unit* (MPU).

The memory types are:

### Normal memory

By default, half of the memory space is classified as Normal memory. Normal memory has many attributes, including Cacheability (Non-cacheable, Write-Through Cacheable, Write-Back Cacheable) and Shareability (Inner Shareability and outer Shareability), that impacts how data can be used in the system. Unaligned accesses to this memory type are allowed. However, under software control, the processor can fault on Unaligned accesses to Normal memory.

### Device memory

Device memory is not *idempotent* and it is generally used by peripherals.

Architecturally, memory locations that are *idempotent* have the following properties:

- Read accesses can be repeated with no side-effects.
- Repeated read accesses return the last value that is written to the resource being read.
- Read accesses can fetch additional memory locations with no side-effects.
- Write accesses can be repeated with no side-effects, if the contents of the location that is accessed are unchanged between the repeated writes or as the result of an exception.
- Unaligned accesses can be supported.
- Accesses can be merged before accessing the target memory system.

For more information, see the *Arm®v8-M Architecture Reference Manual*.

There are restrictions on how Device memory can be ordered, merged, or speculated. These restrictions subdivide Device memory into the following subtypes.

#### Gathering, G and nG

Gathering, G, is the capability to gather and merge requests together into a single transaction. nG represents the non-Gathering attribute.

#### Reordering, R and nR

Reordering, R, is the capability to reorder transactions. nR represents the non-Reordering attribute.

#### Early Write Acknowledgment, E and nE

Early Write Acknowledgment, E, is the capability to accept early acknowledgment of transactions from the interconnect. nE represents the non-Early Write Acknowledgment attribute, indicating that buffering is not permitted. For the Cortex®-M52 processor, nE Device transactions are buffered inside the processor itself. This attribute is then passed to the external interface to ensure that the response is received appropriately.

The Cortex®-M52 processor treats the different types of Device memory identically. However, for MVE instructions, regardless of the Gathering attribute, multiple requests might be merged into one transaction. For Device memory:

- Data accesses are coherent for all system observers.
- All accesses must be aligned to the data type specified in the instruction. Unaligned accesses generate an Alignment fault.

### Remapping

The default memory map defines the Peripheral, External device, *Private Peripheral Bus* (PPB), and Vendor\_SYS regions as Device and the rest of the memory regions as Normal.

- Normal memory can be changed to Device.
- Device memory can be changed to Normal except for the following cases.
  - The PPB region is always Device-nGnRnE.
  - The Vendor\_SYS region is Device-nGnRE and can be changed to Device-nGnRnE.
  - Mapping the Vendor\_SYS region from Device to Normal results in **UNPREDICTABLE** behavior.

For more information on memory types and their attributes, see the *Arm®v8-M Architecture Reference Manual*.

## 8.3 Private Peripheral Bus

The *Private Peripheral Bus* (PPB) memory region provides access to internal and external processor resources.

The following table shows the regions in the memory map where attributes are determined only by the Security state of the processor and cannot be controlled using the SAU or IDAU.

These regions are all associated with either *System Control Space* (SCS) or debug and trace components.



All regions or peripherals listed in the following table contain CoreSight ID registers which are listed in the processor ROM table when the processor is configured to include the region or peripheral.

**Table 8-3: IPPB memory region accesses**

Address Range (inclusive)	Region or peripheral
0xE0000000-0xE000FFFF	<i>Instrumentation Trace Macrocell</i> (ITM), if configured to be included
0xE0001000-0xE0001FFF	<i>Data Watchpoint and Trace</i> (DWT), if configured to be included
0xE0002000-0xE0002FFF	<i>BreakPoint Unit</i> (BPU), if configured to be included
0xE0003000-0xE0003FFF	<i>Performance Monitoring Unit</i> (PMU), if configured to be included
0xE0004000-0xE0004FFF	Reserved
0xE0005000-0xE0005FFF	<i>Reliability, Availability, and Serviceability</i> (RAS) registers
0xE0006000-0xE000DFFF	Reserved
0xE000E000-0xE000EFFF	SCS
0xE000F000-0xE001DFFF	Reserved
0xE001E000-0xE001FFFF	<b>IMPLEMENTATION DEFINED</b> registers <b>Note:</b> The Security state of the processor controls these registers.
0xE0020000-0xE002DFFF	Reserved
0xE002E000-0xE002EFFF	SCS Non-secure alias
0xE003E000-0xE003FFFF	<b>IMPLEMENTATION DEFINED</b> registers Non-secure alias <b>Note:</b> The Security state of the processor controls these registers.

**Table 8-4: EPPB memory region accesses**

Address Range (inclusive)	Region or peripheral
0xE0040000 - 0xE0040FFF	<i>Trace Port Interface Unit</i> (TPIU)
0xE0041000 - 0xE0041FFF	<i>Embedded Trace Macrocell</i> (ETM), if configured to be included

Address Range (inclusive)	Region or peripheral
0xE0042000 - 0xE0042FFF	Cross Trigger Interface (CTI), if configured to be included
0xE0043000 - 0xE0044FFF	Reserved
0xE0045000 - 0xE0045FFF	Embedded Trace Buffer (ETB), if configured to be included
0xE0046000 - 0xE0046FFF	Programmable MBIST Controller (PMC-100)  <b>Note:</b> The PMC-100 contains CoreSight™ ID registers which are listed in the processor ROM table when the processor is configured to include the PMC-100. If the processor is configured with the Security Extension, the PMC-100 can only be programmed by software running in the Secure privileged state, or by the debugger when Secure debug is enabled in the system. The PMC-100 cannot be accessed in Non-secure state.
0xE0047000 - 0xE0047FFF	External Wakeup Interrupt Controller (EWIC), if configured to be included
0xE0048000 - 0xE0048FFF	Software Built-In Self Test (SBIST) controller
0xE0049000 - 0xE00FEFFF	External Private Peripheral Bus (EPPB) APB interface  <b>Note:</b> Peripherals in the EPPB region can apply security checks by using the PPROT[1] signal to determine if the access was made from Secure or Non-secure state and respond with PSLVERR HIGH if the access is not allowed.
MCU level CoreSight™ ROM table base address- (MCU level CoreSight™ ROM table base address +0xFFF)	System-level ROM table  <b>Note:</b> The base address of the system-level ROM table is implementation-dependent.
0xE00FF000 - 0xE00FFFFF	Processor ROM table

## 8.4 Unaligned accesses

The Cortex®-M52 processor has different levels of support for loads and stores to unaligned addresses. Unaligned accesses are less efficient than using aligned memory locations, because the processor must perform a series of transactions to construct the necessary result.

### Non-MVE accesses

For non-MVE accesses the following terminology applies:

#### Access size

The size of the data specified by an instruction.

#### Unaligned access

An access is unaligned if the access size is not aligned with address of the access.

**Table 8-5: Unaligned non-MVE accesses**

Behavior and performance	Non-MVE accesses
Cortex®-M52 processor faulting behavior	Unaligned non-MVE accesses fault in the following scenarios: <ul style="list-style-type: none"> <li>When the access is to the <i>External Private Peripheral Bus</i> (EPPB) region.</li> <li>When the access is to a memory region marked as Device.</li> <li>When the Unaligned trap is enabled (CCR.UNALIGN_TRP=1). For more information on the CCR register, see the <i>Arm®v8-M Architecture Reference Manual</i>.</li> <li>When the access instruction is an <b>LDM</b> or <b>STM</b>.</li> </ul>
Performance implications	Unaligned non-MVE accesses might be result in multiple smaller transfers. Therefore, there is a potential performance impact.

### MVE accesses

For MVE accesses the following terminology applies:

#### Element size

The size of the data specified by an instruction.

#### Unaligned access

An access is unaligned if the element size is not aligned with the address of the access.

**Table 8-6: Unaligned MVE accesses**

Behavior and performance	MVE accesses
Cortex®-M52 processor faulting behavior	Unaligned MVE accesses always raise a UsageFault exception.
Performance implications	If an MVE transaction is not aligned to 32 bits but is still considered to be an aligned MVE transaction, then there is a performance impact because MVE instructions always transfer 128 bits of data as multiple 32 bits data transactions.



### VLDRB, VLDRH, VLDRW examples

To illustrate unalignment in MVE accesses, consider the following Vector Load Register instruction examples:

**Table 8-7: VLDRB, VLDRH, VLDRW examples**

Syntax	Alignment and faulting behavior	Performance implications?
VLDRH.S16 Q0, [R1, #0]	Aligned access	No
VLDRB.S8 Q0, [R0, #1]	Aligned access	Yes
VLDRW.S32 Q0, [R2, #1]	Unaligned access, UsageFault occurs	-



In the preceding examples, the base register values for R1, R0, and R2 are aligned to the data type.

## 8.5 Access privilege level for Device and Normal memory

The AMBA® 5 AXI, AMBA 5 AHB, and AMBA 4 APB protocols include signals that allow the privilege level of an access to be reported to the system.

The Cortex®-M52 processor supports these signals across the Main interface, *Peripheral AHB* (P-AHB), and *External Private Peripheral Bus* (EPPB) interfaces for Device memory. It also supports privilege reporting for Normal memory on P-AHB. However, accesses to Normal memory on Main interface can be buffered and cached so memory read and write requests and instruction fetches from both privileged and unprivileged software can be merged. For these transactions, the AXI signals ARPROT[0] and AWPROT[0], or the AHB signal HPROT[1] are always 1 indicating a privileged access. Access permission to a region of memory can always be restricted to software running in privileged mode by using the *Memory Protection Unit* (MPU).

The *Instruction Tightly Coupled Memory* (ITCM) and *Data Tightly Coupled Memory* (DTCM) interfaces provide signals ITCMPRIV, DOTCMPRIV, and D1TCMPRIV to indicate the privilege of all memory accesses.

For more information on these signals, see the [Instruction Tightly Coupled Memory interface signals](#) and [Data Tightly Coupled Memory interface signals](#).

## 8.6 Memory ordering and barriers

Transactions that are performed on different interfaces can be reordered relative to one another, even if one or more of them is to Device memory.

In this context, the *Internal Private Peripheral Bus* (IPPB) region must be considered as a distinct interface. Therefore, *Private Peripheral Bus* (PPB) accesses can be reordered relative to Device accesses performed on the *Peripheral AHB* (P-AHB) or Main interface.

This is consistent with the architectural memory ordering requirements as defined in the *Arm®v8-M Architecture Reference Manual* based on the assumption that the same peripheral is never mapped onto multiple interfaces.

If stricter ordering is required between two transactions to different interfaces, a *DMB* or *DSB* instruction must be inserted between them. For transactions to the same interface, two transactions to Device memory are always performed in program order.

TCMs are always implicitly Normal memory and any attempt to enforce stricter requirements by changing *Memory Protection Unit* (MPU) attributes are ignored.

The Armv8.1-M architecture includes the load-acquire and store-release instructions. These can be used to implement hardware-level support for the C++11 standard library atomic operations.

ISB instructions are required to guarantee the effect of instructions during context changes because the processor can prefetch several instructions before they are executed.

## 8.7 Execute Only Memory

The Cortex®-M52 processor supports system level use of *eXecute Only Memory* (XOM) on the Main interface and *Tightly Coupled Memory* (TCM) interfaces. The system integrator is responsible for adding relevant system design logic to support use of XOM.

In an XOM configuration, memory that is designated as execute-only cannot be read directly or indirectly by software running on the processor, or by the debugger. XOM operation requires that software is compiled so that literals are constructed through instruction fetches rather than explicit loads from memory. For example, using the *movt* and *movw* instructions.

XOM on the TCM interfaces is supported by the *xTCMASTER* output signal which is set to 0b0000 for instruction fetches from software running on the processor. Any access to an XOM region which is not recognized as an instruction fetch can be aborted by asserting the *xTCMERR* signal. XOM regions protected in this way can never be accessed by *AHB TCM Access* (TCM-AHB) as a read on the slave interface will always result in a TCM access with *xTCMASTER* set to 0b0011.

XOM on the AXI interface requires that instruction fetches can be identified on the AXI interface. This can be done by checking the AXI read ID, *ARPROT*[2] which is only asserted for instruction fetch requests. XOM on the M-AHB interface requires that instruction fetches can be identified on

the AXI interface. This can be done by checking the M-AHB HPROT[0] which is only de-asserted for instruction fetch requests. The processor supports direct access to the cache RAM, therefore, access to the L1 instruction cache must also be restricted. This can be achieved by asserting the external input signal LOCKDCAIC. For code region, unified cache must not be used with XOM because it is not XOM aware design to code region. For non-code region, such as data region, XOM is available for unified cache.

If the PMC-100 is included in the processor configuration, Main interface is not suitable for XOM integration because the internal cache RAMs can always be accessed by on-line *Memory Built-In Self Test* (MBIST). For more information on XOM, see *Arm®v8-M Architecture Reference Manual*.

## 9. Memory Authentication

This chapter describes the *Memory Authentication Unit* (MAU) responsible for controlling access to memory.

### 9.1 MAU features

The *Memory Authentication Unit* (MAU) receives requests from units that perform memory accesses, and the MAU returns responses to these units. These responses are a combination of all the responses from the *Memory Protection Unit* (MPU), *Security Attribution Unit* (SAU), *Implementation Defined Attribution Unit* (IDAU), and *TCM Gate Unit* (TGU). The MAU contains the following units or interfaces to units.

- MPU. For more information, see the [Memory Protection Unit](#).
- TGU. For more information, see the [TCM Gate Units](#).
- SAU. For more information, see the [Security Attribution Unit](#).
- Interface to the IDAU. For more information, see the [Implementation Defined Attribution Unit](#).
- Interface to the *Load Store Unit* (LSU) from the MAU. The LSU makes MAU lookup requests for loads, stores, and *Preload Data* (PLD), linefills, stacking, and unstacking.
- Interface to the TCU from the TGU. The TCU makes TGU requests through the *AHB TCM Access* (TCM-AHB) interface for *Direct Memory Accesses* (DMAs). For more information, see the [TCM interfaces](#).
- Interface to the *Instruction Fetch Unit* (IFU) from the MAU. The IFU makes lookup requests for instructions and vector fetches.



When changing security attribution of an address by either reprogramming the SAU or changing the external IDAU mappings, cache maintenance is required.

### 9.2 Security Attribution Unit

The optional *Security Attribution Unit* (SAU) provides security attribution for the Cortex®-M52 processor.

#### SAU features

- The SAU is a programmable unit that determines the security of an address.
- It is only implemented if the Security Extension is included in the processor.
- The number of regions that are included in the SAU can be configured in the Cortex®-M52 implementation to be 0, 4, or 8.

- The SAU is not used for *AHB TCM Access* (TCM-AHB) accesses.

## Exemptions and faults

- The *System Control Space* (SCS) and all debug components are exempt from security checking.
- Accesses that violate the security settings cause a SecureFault. In this case, any potential MemManage Fault is masked and the access on the bus is blocked.
- SecureFaults do not prevent Speculative accesses to the caches or TCMs, however, an access that faults never updates processor state.

## Enabling the SAU

The SAU\_CTRL.ENABLE determines whether programming the SAU affects the security of an address. For the Cortex®-M52 processor, this value resets to 0.

### 9.2.1 SAU register summary

The *Security Attribution Unit* (SAU) has various registers that are associated with its function.

Each of these registers is 32 bits wide. The following table shows the SAU register summary. See the *Arm®v8-M Architecture Reference Manual* for more information about the register addresses, access types, and reset values. All the registers in the following table are not banked between Security states.

**Table 9-1: SAU register summary**

Address	Name	Type	Reset value	Description
0xE000EDD0	SAU_CTRL	RW	0x00000000	SAU Control Register
0xE000EDD4	SAU_TYPE	RO	0x0000000X <b>Note:</b> SAU_TYPE[3:0] depends on the number of SAU regions included. This value can be 0, 4, or 8.	SAU Type Register
0xE000EDD8	SAU_RNR	RW	0x000000XX	SAU Region Number Register
0xE000EDDC	SAU_RBAR	RW	0xFFFFFFFF0	SAU Region Base Address Register
0xE000EDE0	SAU_RLAR	RW	<b>UNKNOWN</b> , bit[0] resets to 0.	SAU Region Limit Address Register
0xE000EDE4	SFSR	RW	0x00000000	Secure Fault Status Register
0xE000EDE8	SFAR	RW	<b>UNKNOWN</b>	Secure Fault Address Register

## 9.2.2 Security levels

The security level that the SAU returns is a combination of the region type that is defined in:

- The internal SAU, if configured to be included
- The associated external *Implementation Defined Attribution Unit* (IDAU)

The final security level uses the higher security level indicated by the SAU or IDAU.

When the SAU\_CTRL.ENABLE is zero, the default internal security levels is selected by the SAU\_CTRL.ALLNS field. In the Cortex®-M52 processor, the SAU\_CTRL register resets to zero, setting all memory (apart from some specific regions in the PPB space) to Secure, and preventing any override of the security level by the IDAU.

The following table shows examples of how the final security level is chosen.

**Table 9-2: Final security level selection examples**

IDAU	SAU	Final security
Secure	Secure, Non-secure, or Non-secure Callable	Secure
Secure, Non-secure, or Non-secure Callable	Secure	Secure
Non-secure Callable or Non-secure	Non-secure Callable	Non-secure Callable
Non-secure Callable	Non-secure Callable or Non-secure	Non-secure Callable
Non-secure	Non-secure	Non-secure

For more information on the IDAU, see [Implementation Defined Attribution Unit](#).

## 9.3 Memory Protection Unit

The Cortex®-M52 processor supports Arm *Protected Memory System Architecture* (PMSA). The *Memory Protection Unit* (MPU) is an optional component that is primarily used for memory region protection.

### MPU features

The MPU features include:

- Memory region protection.
- Access permissions.
- Exporting memory attributes to the system.
- The MPU is not used for *AHB TCM Access* (TCM-AHB) accesses.
- You can use the MPU to:
  - Enforce privilege rules.
  - Separate processes.
  - Manage memory attributes.

## Permission and access violations

MPU mismatches and permission violations invoke the MemManage Fault handler. These violations result in MemManage Faults and the access on the bus is blocked. For more information on MemManage Faults, see the *Arm®v8-M Architecture Reference Manual*. MemManage Faults do not prevent Speculative accesses to the caches or TCMs, however, an access that faults never updates processor state.

## MPU configuration

The MPU can be configured to support 0, 4, 8, 12, or 16 memory regions.

If the Security Extension is included in the Cortex®-M52 processor, memory protection can be duplicated between Secure and Non-secure MPU (MPU\_S and MPU\_NS).

The number of regions in the Secure and Non-secure MPU can be configured independently, and each can be programmed to protect memory for the associated Security state.

### 9.3.1 Memory Protection Unit register summary

The *Memory Protection Unit* (MPU) has various registers that are associated with its function.

Each of these registers is 32 bits wide. If the MPU is not present in the implementation, then all of these registers *Read-As-Zero* (RAZ). The following table shows the MPU register summary.

Each of these registers is 32 bits wide. The following table shows the MPU register summary.

See the *Arm®v8-M Architecture Reference Manual* for more information about the register addresses, access types, and reset values. All the registers in the following table are banked between Security states.

**Table 9-3: MPU register summary**

Address	Name	Type	Reset value	Description
0xE000ED90	MPU_TYPE	RO	0x0000xx00 <b>Note:</b> MPU_TYPE[12:8] depends on the number of MPU regions configured. This value can be 0, 4, 8, 12, or 16.	MPU Type Register
0xE000ED94	MPU_CTRL	RW	0x00000000	MPU Control Register
0xE000ED98	MPU_RNR	RW	0x0000000X	MPU Region Number Register
0xE000ED9C	MPU_RBAR	RW	UNKNOWN	MPU Region Base Address Register
0xE000EDA0	MPU_RLAR	RW	UNKNOWN, bit [0] resets to 0.	MPU Region Limit Address Register
0xE000EDA4	MPU_RBAR_A1	RW	UNKNOWN	MPU Region Base Address Register Alias 1

Address	Name	Type	Reset value	Description
0xE000EDA8	MPU_RLAR_A1	RW	<b>UNKNOWN</b> , bit[0] resets to 0.	MPU Region Limit Address Register Alias 1
0xE000EDAC	MPU_RBAR_A2	RW	<b>UNKNOWN</b>	MPU Region Base Address Register Alias 2
0xE000EDB0	MPU_RLAR_A2	RW	<b>UNKNOWN</b> , bit[0] resets to 0.	MPU Region Limit Address Register Alias 2
0xE000EDB4	MPU_RBAR_A3	RW	<b>UNKNOWN</b>	MPU Region Base Address Register Alias 3
0xE000EDB8	MPU_RLAR_A3	RW	<b>UNKNOWN</b> , bit[0] resets to 0.	MPU Region Limit Address Register Alias 3
0xE000EDC0	MPU_MAIRO	RW	<b>UNKNOWN</b>	MPU Memory Attribute Indirection Register 0
0xE000EDC4	MPU_MAIR1	RW	<b>UNKNOWN</b>	MPU Memory Attribute Indirection Register 1

## 9.4 Implementation Defined Attribution Unit

The Cortex®-M52 processor supports an external *Implementation Defined Attribution Unit* (IDAU) to allow the system to determine the security level that is associated with any given address.

- The processor has two external interfaces for the IDAU with identical signals, properties, and requirements.
  - An interface for instruction fetches and exception vector read operations.
  - An interface for all other data read and write operations from load and store instructions, register stacking on exception entry and exit, and debug memory accesses.
- The IDAU (and SAU) is not used for *AHB TCM Access* (TCM-AHB) accesses.

### Security levels

The security level that the *Memory Authentication Unit* (MAU) returns is a combination of the region type defined in the internal SAU, if configured to be included, and the security type from the IDAU. For more information, see [Security Attribution Unit](#).

### 9.4.1 IDAU interface and backwards compatibility

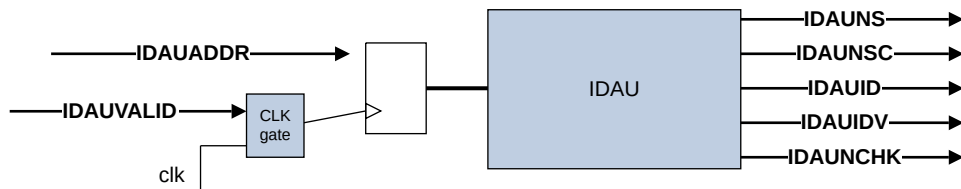
The *Implementation Defined Attribution Unit* (IDAU) interface protocol in the Cortex®-M52 processor has a two-stage pipeline, allowing lookup, comparator, and resulting multiplexed logic



to be balanced across a register slice to balance timing according to **IMPLEMENTATION-SPECIFIC** requirements.

The following figure shows how backwards compatibility can be implemented to allow for use with existing IDAU system designs.

**Figure 9-1: Cortex®-M52 IDAU interface backward compatibility**



To optimize your design, Arm® recommends that the external IDAU is implemented with the processor logic to allow EDA tools to balance the timing of the IDAU logic with the internal *Security Attribution Unit* (SAU).

## 9.5 Security attribution signals

Security attribution is indicated for the Cortex®-M52 interfaces on the following signals:

- Bit [1] of ARPROT and AWPROT for the *AXI Main* (M-AXI) interface.
- HNONSECC and HNONSECSYS for *AHB Main* (M-AHB) interface.
- HNONSECP for the *Peripheral AHB* (P-AHB) interface.
- HNONSECD for the *Debug AHB* (D-AHB) interface.
- HNONSECS for the *AHB TCM Access* (TCM-AHB) interface.

Using these signals ensures that the relevant interface components prevent Non-secure transfers to Secure memory or peripherals.



- TCM-AHB requests do not use the SAU and IDAU for security checking. However, HNONSECS is taken into consideration for security access gating using the *TCM Gate Unit* (TGU). See [Security access gating using the TGU](#).
- The security attribute depends on address of the location being accessed, and not on the Cortex®-M52 processor Security state that executes the load/store instructions or *Debug Access Port* (DAP) Security state that generates the debug request.
- Permitted DAP accesses to Secure *System Control Space* (SCS) registers in the range 0xE000E000-0xE000EFFF are affected by the value of the following:

- Secure debug enabled bit in the Debug Halting Control Status Register, DHCSR.S\_SDE
- Secure banked register select enable bit in Debug Security Control and Status Register, DSCSR.SBRSELEN
- Secure banked register select bit in Debug Security Control and Status Register, DSCSR.SBRSEL
- Current security state of the processor.

**Table 9-4: DAP accesses to Secure SCS registers**

DHCSR.S_SDE	DSCSR.SBRSELEN	DSCSR.SBRSEL	Current Security state of the processor	View of the register accessed
0	-	-	-	Non-secure
1	0	-	Non-secure	Non-secure
1	0	-	Secure	Secure
1	1	0	-	Non-secure
1	1	1	-	Secure

For more information on DHCSR and DSCSR, see the *Arm®v8-M Architecture Reference Manual*.

## 9.6 TCM Gate Units

There are two *TCM Gate Units* (TGUs), one for *Instruction Tightly Coupled Memory* (ITCM) accesses (ITGU), and one for *Data Tightly Coupled Memory* (DTCM) accesses (DTGU), that are responsible for TCM security gating and control.

For more information on how the TGUs are responsible for security access control, see [TCM and P-AHB security access control](#).

## 9.7 TCM and P-AHB security access control

The Cortex®-M52 processor provides a mechanism to support further or a more fine-grained security access control on the TCM and *Peripheral AHB* (P-AHB) interfaces than provided by the SAU and IDAU.

This mechanism is compatible with the external gating mechanism described in *Arm® Platform Security Architecture Trusted Base System Architecture for Arm®v6-M, Arm®v7-M, and Arm®v8-M*.

To achieve additional security access control, you must use memory aliasing, configure the *Implementation Defined Attribution Unit* (IDAU) or *Security Attribution Unit* (SAU), and implement security gating.

## Memory aliasing and IDAU and SAU configuration

Memory aliasing can be applied to the TCM and P-AHB interfaces. Memory aliasing is a duplication of all memory-mapped components in Secure and Non-secure address regions. These regions must be defined as Secure and Non-secure using the IDAU or SAU. For more information, see [Memory aliasing and IDAU/SAU configuration](#).

For more information on the SAU and IDAU, see [Security Attribution Unit](#) and [Implementation Defined Attribution Unit](#) respectively.

### Security gating

The *TCM Gate Unit* (TGU) provides security gating for TCM accesses only. For more information on TGU security gating, see [Security access gating using the TGU](#). To implement memory aliasing with the TCMs, you must use the TGU to maximize the benefits of the additional level of security that it provides.

If memory aliasing is not enabled (using the CFGMEMALIAS signal), the TGU is not used.

Accesses to the P-AHB require you to include your own external security gating logic.



Note

Additionally, memory aliasing can be done for the Main interface, and all gating must be implemented externally. Therefore, the description of this behavior is outside the scope of this document. For more information, see the *Arm® Platform Security Architecture Trusted Base System Architecture for Arm®v6-M, Arm®v7-M, and Arm®v8-M*.

## 9.7.1 Memory aliasing and IDAU/SAU configuration

In normal operation, the TCM and *Peripheral AHB* (P-AHB) interfaces are mapped to regions in the memory map.

<b>Code region</b>	Base address 0x00000000 is used for <i>Instruction Tightly Couple Memory</i> (ITCM).
<b>SRAM region</b>	Base address 0x20000000 is used for <i>Data Tightly Couple Memory</i> (DTCM).
<b>Peripheral region</b>	Base address 0x40000000 is used for P-AHB.



Note

The TCM base address is configurable. For more information about the configurable TCM base address, see [TCM configuration](#).

The TCM regions extend from their base to a limit that is defined by the physical size (in bytes) of the TCM set by the input signals CFGITCMSZ and CFGDTCMSZ. The P-AHB region extends from the base to its region size (in bytes) defined by the CFGPAHBSZ input signal.

Memory aliasing is enabled by tying the external input signal CFGMEMALIAS[2:0] to a non-zero value. The aliased address bit can be set from bit [26] to bit [28] using the CFGMEMALIAS[2:0] signal. The address bit that is used for memory alias is determined by the following options:

- 0b001, indicating that the alias bit is bit[26].
- 0b010, indicating that the alias bit is bit[27].
- 0b100, indicating that the alias bit is bit[28].

This results in:

- A second CODE and SRAM address region mapped to the ITCM and DTCM respectively.
- A second region in the Peripheral region to be mapped to the P-AHB interface.

0b000 indicates that there is no memory aliasing. Setting the address bit to any other value results in **UNPREDICTABLE** behavior.

For example, if you are using CFGMEMALIAS[2:0] for memory aliasing and you have set CFGMEMALIAS[2:0] to 0b100 (bit [28] is used as the alias bit), CFGPAHBSZ should correspond to the actual size of the P-AHB region (in bytes):

- The base address of the P-AHB region is from 0x40000000-0x40000000+size\_in\_bytes(CFGPAHBSZ)-1
- The alias address of the P-AHB region is from 0x50000000-0x50000000 + size\_in\_bytes(CFGPAHBSZ)-1.

The following table demonstrates an example of memory aliasing for the ITCM, DTCM, and P-AHB when the alias is configured for bit[28] of the address. The actual accessible TCM regions depend on the size of the TCM configured in the processor. In the following table, the size of the P-AHB region is limited to 256MB to avoid overlap with the alias at bit[28].

**Table 9-5: Example TCM memory address aliasing**

Address	Target region
0x00000000-0x00FFFFFF	ITCM
0x10000000-0x10FFFFFF	ITCM alias
0x20000000-0x20FFFFFF	DTCM
0x30000000-0x30FFFFFF	DTCM alias
0x40000000-0x4FFFFFFF	P-AHB
0x50000000-0x5FFFFFFF	P-AHB alias



Note

Base and alias regions can overlap in the Peripheral region because the P-AHB interface can be mapped to the entire 512MB. However, Arm recommends that you avoid doing this because the behavior is **UNPREDICTABLE**. The aliasing logic only affects the target interface for P-AHB and TCM and it does not change the actual address. External security logic on this interface must mask the address accordingly to map the two aliased addresses to the same physical peripheral.

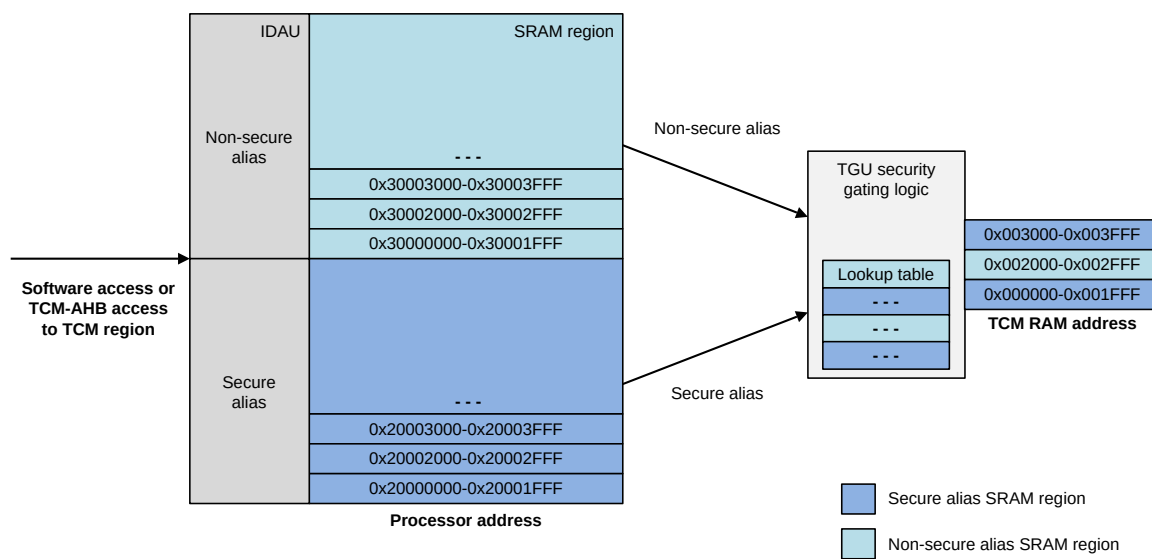
## IDAU/SAU configuration for security access control

When memory aliasing is enabled, the *Implementation Defined Attribution Unit* (IDAU) or *Security Attribution Unit* (SAU) must be set up to map the two alias regions for each interface. This allows one region to be mapped as Secure and the other region to be mapped as Non-secure. This Secure and Non-secure mapping guarantees that software can access any given physical address in the TCM or P-AHB through external address mapping as either Secure or Non-secure regions.

For more information on setting up the IDAU using the relevant IDAU signals, see [IDAU interface signals](#).

The following figure shows an example configuration of memory aliasing and IDAU configuration in the SRAM region and the DTCM using bit [28] of the address.

**Figure 9-2: Example security alias and gating configuration on the DTCM**



## 9.7.2 Security access gating using the TGU

The *TCM Gate Unit* (TGU) is a security gate that allows the security attribute of a *Tightly Coupled Memory* (TCM) access to be checked against the security mapping for the address.

There are two optional TGUs, one for the *Instruction Tightly Coupled Memory* (ITCM) and one for the *Data Tightly Coupled Memory* (DTCM).

Each TCM is divided into blocks and a TGU lookup table is used to lookup the security mapping for an address. This is done in either of the following ways:

- For software accesses, the security mapping from the TGU lookup table is checked against the security attribute from the *Security Attribution Unit* (SAU) and *Implementation Defined Attribution Unit* (IDAU).
- For TCM-AHB accesses, the security mapping from the TGU lookup table is checked against the HNONSECS input signal which provides security level information for TCM-AHB accesses.

## 9.7.3 TGU configuration

Each *TCM Gate Unit* (TGU) is configured using the `xTGU`, `xTGUBLKSZ`, and `xTGUMAXBLS` parameters.



In this section, `xTGU` refers to *Instruction TCM Gate Unit* (ITGU) and *Data TCM Gate Unit* (DTGU).

The `xTGU` parameter configures the inclusion of the ITGU or DTGU, the `xTGUBLKSZ` parameter determines the block size, and `xTGUMAXBLS` determines the maximum number of available blocks (which in turn defines the number of physical registers included in the TGU logic). The processor supports up to a maximum of 512 blocks for each TGU.

The `xTGUMAXBLS` parameter is provided to allow a single processor implementation to support security gating across multiple different TCM size configurations using the external input signals `CFGITCMSZ` and `CFGDTCMSZ`.



You must configure `xTGUMAXBLS` and `xTGUBLKSZ` to match the required range of TCM size. A TGU configuration is valid if both of the conditions in the following table are met.

**Table 9-6: TGU configuration conditional validity**

Condition	Formula
Block size * Maximum number of blocks = Maximum physical size of the TCM	$xTGUBLKSZ \times xTGUMAXBLS = CFGxTCMSZ_{max} + 4$
Block size < Minimum physical size of the TCM	$xTGUBLKSZ < CFGxTCMSZ_{min} + 4$

This ensures that there are enough blocks to cover the largest TCM size and that at least two blocks cover the minimum TCM size. If these parameters are configured incorrectly, the TGU behavior becomes **UNPREDICTABLE**.

For a given processor implementation and integration, reading the xTGU\_CFG.NUMBLKS and xTGU\_CFG.BLKSZ register bitfields determines the number of available blocks in the lookup table and the block size respectively. For more information on these registers, see [ITGU\\_CFG and DTGU\\_CFG, ITGU and DTGU Configuration Registers](#).

When TCM gating is enabled, the Code and SRAM region of the processor memory map is aliased so that two regions map onto the same physical TCM address. These two regions should be mapped to different security levels. The security level attributed to the logical address used by software is always used to control the TGU. The two alias regions always map to the same physical address in the TCM memory.

The following table shows an example configuration where the processor ITGU is configured with 1KB blocks and supports a maximum ITCM size of 64KB and a minimum ITCM size of 4KB. In this case, ITGUMAXBLKS must be configured to 0b0110 or 64 blocks.

**Table 9-7: Example TGU configuration for 1KB block size**

ITCM size	CFGITCMSZ	ITGUBLKSZ	ITGUMAXBLKS	ITGU_CFG.NUMBLKS	ITGU_CFG.BLKSZ
4KB	0b0011	0b0101	0b0110	0b0010	0b0101
8KB	0b0100	0b0101	0b0110	0b0011	0b0101
16KB	0b0101	0b0101	0b0110	0b0100	0b0101
32KB	0b0110	0b0101	0b0110	0b0101	0b0101
64KB	0b0111	0b0101	0b0110	0b0110	0b0101

### TGU block lookup table

Each block entry in the lookup table can be accessed by software using the xTGU\_LUTn registers. Each register contains up to 32 block entries. For a valid block, the entry bit determines the required security level. All blocks reset to 0, therefore, at reset, all TCM memory is considered as Secure.

Any unused block entries in the lookup table, due to the configuration, do not affect the operation of the security gate and the corresponding xTGU\_LUTn bitfield is RAZ/WI when accessed by software.

### TGU enable and locking

TCM gating is enabled by tying the external input signal CFGMEMALIAS to a non-zero value.

The TGU can be locked from software access using the external input signals LOCKITGU and LOCKDTGU. When these signals are asserted the corresponding TGU registers become read-only. This allows a TGU configuration to be programmed and then locked from further changes by software.



## 9.7.4 Security check and fault response

Accesses to a memory region that the TGU protects only proceed if the security level of the request matches the programmed security of the block. At reset, all blocks are Secure.

- Read requests on the external TCM interfaces are always Speculative, regardless of whether the access passes the security check in the TGU. Data from the RAM is always ignored if the check fails and the processor state is never updated.
- If the security check fails, write requests are always ignored and never carried out on the TCM interface.

The result of a security check mismatch in the TGU depends on the type of the access and the configuration of the ITGU\_CTRL or DTGU\_CTRL registers. The access is either ignored or generates a fault:

- A security check mismatch on an instruction fetch always results in a BusFault. The fault is recorded in AFSR.FTGU.
- If ITGU\_CTRL.DBFEN or DTGU\_CTRL.DBFEN is set, a security check mismatch on a data read or write results in a precise BusFault. The fault is recorded in AFSR.PTGU. If ITGU\_CTRL.DBFEN or DTGU\_CTRL.DBFEN is not set, no exception is raised.
- If ITGU\_CTRL.DEREN or DTGU\_CTRL.DEREN is set, a security check mismatch on a read or write to the TCM from the TCM-AHB signals an error on the interface. For read and write transactions, an error is also returned on the TCM-AHB signal. For all mismatched read accesses, zero is returned to prevent any leaks of Secure data and mismatched write accesses are ignored.



If a data read access on the TCM returns an error on the interface (ITCMERR or DTCMERR input signal is asserted) for an address which fails the TGU security check and ITGU\_CTRL.DBFEN or DTGU\_CTRL.DBFEN is not set, then the overall behavior is RAZ/WI instead of raising a BusFault. This is consistent with a security fault response from the *Memory Authentication Unit* (MAU).

---

# 10. Memory system

This chapter describes the Cortex®-M52 processor memory system.

## 10.1 Memory system features

The Cortex®-M52 processor memory system is an interface between the processor core and the cache RAMs, external memory interfaces and memory-mapped registers.



For more information on how these units and interfaces interact with each other, see the [Cortex-M52 processor has fixed and optional component blocks](#).

### Load Store Unit

The *Load Store Unit* (LSU) receives load and store accesses from the *Data Processing Unit* (DPU) and distributes these requests to the correct unit and returns any data or responses to the DPU. The LSU contains the *Peripheral Interface Unit* (PIU) which handles all the loads and stores to internal and external peripherals.

### Peripheral Interface Unit

The *Peripheral Interface Unit* (PIU) is responsible for the handling of loads or stores from the LSU to peripheral units *External Private Peripheral Bus* (EPPB), *Internal Private Peripheral Bus* (IPPB), and P-AHB.

### TCM Control Unit

The *TCM Control Unit* (TCU) arbitrates requests between the LSU and *Instruction Fetch Unit* (IFU), accesses the TCMs, and returns any data or responses to the requesting unit. The TCU contains a write queue for *AHB TCM Access* (TCM-AHB) writes and a read prefetcher to improve the performance of 32-bit and 64-bit incrementing reads.

The TCU contains a buffer for software stores to the TCM.

### Tightly Coupled Memories

The Cortex®-M52 processor has two TCM memory types, the *Instruction Tightly Coupled Memory* (ITCM) and *Data Tightly Coupled Memory* (DTCM). One ITCM interface and two DTCM interfaces (D0TCM and D1TCM) exist.

All the TCM interfaces are 39 bits wide (32 bits for data and 7 bits for *Error Correcting Code* (ECC)).

ECC generation and correction logic can optionally be included for each TCM interface and an ECC error indication interface.

Memory accesses to the TCM, required for fetching instructions and for data transfer instructions, are performed if the address is in an enabled TCM region. Accesses that are not serviced by the TCM region are passed through the Main interface or one of the peripheral interfaces.

## Data Cache Unit

The *Data Cache Unit* (DCU) contains a four-way set-associative data cache and handles all accesses to this cache. These accesses include loads, stores, cache maintenance operations, evictions, and ECC error detection and correction.

The DCU can be configured to include logic to detect and process ECC errors.

## Instruction Cache Unit

The *Instruction Cache Unit* (ICU) contains a two-way set-associative instruction cache, and it accepts instruction fetch requests from the IFU and returns data from either the instruction cache, the linefill buffer, or the BIU.

The ICU can be configured as unified cache. It is still a two-way set-associative cache. The ICU accepts instruction fetch requests from the IFU and load requests from LSU, returns data from the unified cache, the linefill buffer, or the BIU.

The ICU can be configured to include logic to detect and process ECC errors.

## Store Buffer

The *Store Buffer* (STB) has four 32-bit slots that buffer stores to the Main interface bus.

- For Cacheable stores, the STB sends a lookup request to the DCU to see if the target address is in the cache. If it is, then the data is written directly to the cache. If the target address is not in the cache and the access has a Write-Allocate hint, then the DCU makes a linefill request to the *Bus Interface Unit* (BIU) and writes the data into the BIU linefill buffer. If the target address is not in the cache and it does not have a Write-Allocate hint, then the store is written out to the Main interface bus.
- For Non-cacheable data, the data is written to the BIU write buffer.
- Write-Through stores are written out to the Main interface bus even if they have been written into the cache.

## Bus Interface Unit

The BIU contains one 16-byte write buffer and one 32-byte linefill buffers.

The BIU coordinates the following accesses to the Main interface.

- Loads from the LSU (or from ICU in unified cache)
- Stores from the STB
- Evictions from the DCU
- Fetches from the IFU
- Linefills triggered by PLD instructions

Non-cacheable loads go directly to the Main interface bus. Stores are buffered internally with the intention of being combined in a burst on the AXI. However, stores are not buffered when M-AHB is used. Cacheable Read-Allocate loads and Cacheable Write-Allocate stores trigger linefills and the data from the Main interface bus is buffered in the linefill buffer until the line is complete and it can be allocated in the DCU.

The linefill buffers also buffer load data from Non-cacheable bursts.

## MBIST Interface Unit

The *MBIST Interface Unit* (MIU) provides the *Memory Built-In Self Test* (MBIST) interface.

The MBIST interface supports on-line and production MBIST.

### **M-AHB interface**

The M-AHB interface is two 32 bits wide AHB interfaces, and connects to the external memory system, code region and system region.

### **M-AXI interface**

The M-AXI interface is 32 bits wide and connects to the external memory system.

### **Peripheral-AHB interface**

The PIU includes a 32-bit *Peripheral-AHB* (P-AHB) interface for accessing external peripherals.

### **AHB TCM Access (TCM-AHB) interface**

The TCM-AHB interface is 32 bits wide and allows system accesses in and out of the TCMs.

### **PPB interfaces**

The PIU includes the Internal Private Peripheral Bus (IPPB) interface to access internal PPB registers, and the External PPB (EPPB) APB interface to access external PPB registers.

## **10.2 Memory system faults**

Memory system faults can occur on instruction fetches and data accesses.

Faults can occur on instruction fetches for the following reasons:

- *Memory Protection Unit* (MPU) MemManage fault.
- *Security Attribution Unit* (SAU) or *Implementation Defined Attribution Unit* (IDAU) SecureFault.
- BusFaults that are caused by an external AXI slave error (SLVERR), an external AXI decode error (DECERR), or corrupted transactions (RPOISON).
- BusFaults that are caused by an external M-AHB response error.
- TCM external error.
- Uncorrectable *Error Correcting Code* (ECC) errors in the TCM.
- Breakpoints and vector catch events.
- *TCM Gate Unit* (TGU) faults.

Faults can occur on data accesses for the following reasons:

- MPU MemManage fault.
- Alignment UsageFault.
- SAU or IDAU SecureFault.
- BusFaults that are caused by an external AXI slave error (SLVERR), an external AXI decode error (DECERR), or corrupted read data (RPOISON).
- BusFaults that are caused by an external M-AHB response error.
- BusFaults because of errors on the *External Private Peripheral Bus* (EPPB) APB interface.
- External AHB error from the *Peripheral-AHB* (P-AHB) interface.

- TCM external error.
- Uncorrectable ECC errors in the TCM or L1 data cache.
- Watchpoints.
- *M-profile Vector Extension* (MVE) transactions, stacking, or unstacking to the *Private Peripheral Bus* (PPB) space.
- TGU faults.
- Unprivileged accesses to system registers which only privileged code can access.

## 10.2.1 Classes of fault

Faults can be classified as MemManage Faults, BusFaults, SecureFaults, and UsageFaults.

### MemManage faults

The *Memory Protection Unit* (MPU) can generate a fault for various reasons.

For more information on MemManage Faults, see [Permission and access violations](#).

### Bus faults

A memory access or instruction fetch performed through the *AXI Main* (M-AXI) interface can generate different types of responses:

- Slave error (SLVERR).
- Decode error (DECERR).

AXI bus errors cause precise or imprecise BusFaults. Also, if the AMBA® 5 AXI signal, RPOISON, is asserted, an AXI read can generate a BusFault.

A memory access performed through the *Peripheral AHB* (P-AHB) interface can generate a single error response. The processor manages this in the same way as a response of SLVERR from the AXI interface.

Whether a memory or instruction fetch access on the TCM interface can be performed or not relies on the *TCM Gate Unit* (TGU), if implemented. Depending on the programming of the TGU, TGU faults can generate errors.

- For loads or stores, errors cause synchronous BusFaults.
- For read and write accesses from the *AHB TCM Access* (TCM-AHB) interface, an error causes a TCM-AHB error response on HRESPS. For writes, only TCM interface errors on ITCMERR or DTCMERR result in an imprecise error response on TCM-AHB through SAHBWABORT.

Synchronous BusFaults are generated in the following cases:

- Instruction fetches.
- Data loads.
- Stores that generate a TGU fault.

- Stores to PPB that cause a privilege violation.
- *M-profile Vector Extension* (MVE) stores and stacking to the PPB space.
- Uncorrectable *Error Correcting Code* (ECC) errors.

Asynchronous BusFaults are generated in the following cases:

- All stores except those that generate synchronous BusFaults.
- Dirty linefills that cause an M-AXI/M-AHB bus error.
- Data cache eviction or cache maintenance cause uncorrectable ECC errors.

## Secure Faults

If accesses do not pass the security attribution checks that the Memory Authentication Unit (MAU) performs, then a SecureFault is raised.

For more information on security attribution, see [Memory Authentication](#).



Note

In most of the memory regions, debugger accesses are subject to validation and attribution. That is, the final Security state of an access on the *AXI Main* (M-AXI) interface, indicated on ARPROT[1] and AWPROT[1] signals, the final Security state of an access on the *AHB Main* (M-AHB) interface, indicated on HNONSECC and HNONSECSYS signals, the *Peripheral AHB* (P-AHB) interface, indicated on HNONSECP signal, or the *External Private Peripheral Bus* (EPPB) APB interface, indicated on PPROT[1] signal, is set by the *Security Attribution Unit* (SAU) in the same way as software generated accesses. The SAU blocks memory accesses which do not have the required permissions. For example, accesses to memory marked as Secure in the SAU when DHCSR.S\_SDE is 0 or HNONSECD is HIGH. This results in an error response on the *Debug AHB* (D-AHB) interface, but unlike accesses that originate from software, a SecureFault is not raised.

## Usage faults

UsageFault exceptions occur in the following cases:

- Any unaligned access when CCR.UNALIGN\_TRP is set results in an UNALIGNED UsageFault exception. For more information on CCR, see the *Arm®v8-M Architecture Reference Manual*.
- Architecture ReferenceUnaligned accesses to Device memory regions are not supported and result in an UNALIGNED UsageFault exception.
- Unaligned accesses from an instruction that does not support unaligned accesses result in an UNALIGNED UsageFault exception. For more information on these instructions, see the *Alignment behavior* section in the *Arm®v8-M Architecture Reference Manual*.
- For *M-profile Vector Extension* (MVE) operations, a load or store access is considered unaligned if the address is not aligned to the specified element size. Using an address for an MVE load or store which is not aligned to the element size results in an UNALIGNED UsageFault being raised. For more information on MVE and elements, see the *Arm®v8-M Architecture Reference Manual*.
- Accessing a coprocessor that does not exist results in a NOCP UsageFault.



For more information on external coprocessors, see [External coprocessors](#). Also, for more usage restriction information, see [Usage restrictions](#).

## 10.3 Memory system behavior

The behavior of the memory system depends on the type attribute of the memory that is being accessed. Only Normal, cacheable memory regions can be cached in the RAMs.

The following items and the table that follows summarize the memory types and their associated memory system behavior:

- The memory system supports all memory types specified in the *Arm®v8-M Architecture Reference Manual*.
- For the data cache, all Shareable transactions are forced to be Non-cacheable because the Cortex®-M52 processor must be data coherent with other observers in the Shareability domain. On the data side, if a transaction is marked as Non-shareable, then caching can occur if the data cache is enabled (CCR.DC=1) and active (MSCR.DCACTIVE=1). For more information on CCR, see the *Arm®v8-M Architecture Reference Manual*. For more information on MSCR, see [MSCR, Memory System Control Register](#).
- For the instruction cache, transactions marked as Shareable Cacheable are not forced to be Non-cacheable because the instruction cache cannot be dirty and its contents are always consistent with the external memory. Unless, the external memory changes, in which case, the instruction cache is invalidated. Therefore, caching occurs irrespective of the Shareability attribute. On the instruction side, caching can occur if, the instruction cache is enabled (CCR.IC=1) and active (MSCR.ICACTIVE=1). For more information on CCR, see the *Arm®v8-M Architecture Reference Manual*. For more information on MSCR, see [MSCR, Memory System Control Register](#). The processor caches Shareable Cacheable instruction fetches. Therefore, instruction cache software maintenance is always required for self-modifying code because only data access coherency is supported.
- The unified cache is a read-only cache. However, the unified cache is not a typical unified cache of Arm Architecture. A unified cache can be included to optimize the performance of a flash memory located in Code region.
  - Only instruction and data load accesses to the Code region of the memory map can be cached. Store data cannot be cached in unified cache.
  - The transactions of data side marked as Shareable Cacheable are forced to be Non-cacheable.
  - The transactions of instruction side marked as Shareable Cacheable are Not forced to be Non-cacheable. If the external shared memory changes, the unified cache will be invalidated by software maintenance.
  - The unified cache inherits all relevant registers of data cache for both instruction and data side accesses.

- On the data and instruction side, caching can occur only if, the unified cache is enabled (CCR.DC=1) and active (MSCR.DCACTIVE=1). For more information on CCR, see the *Arm®v8-M Architecture Reference Manual*. For more information on MSCR, see [MSCR, Memory System Control Register](#).
- The processor caches Shareable Cacheable instruction fetches. Therefore, unified cache software maintenance is always required for self-modifying code because only data access coherency is supported.
- To keep data coherency, store to an address cached in unified cache will automatically invalidate the cache line of unified cache.
- The store buffer supports all stores to Main interface. It also handles the special behavior required for no Write-Allocate mode.
- All Shareable exclusive transactions to the Main interface and *Peripheral AHB* (P-AHB) interfaces are marked as exclusive.
- All Non-shareable exclusive transactions to the Main interface and P-AHB interfaces are not marked as exclusive.
- Only Normal memory is considered idempotent. For more information on the properties of idempotent Normal memory, see the *Normal memory* section *Arm®v8-M Architecture Reference Manual*.
- For exclusive accesses to Non-shared memory only the internal exclusive monitor is updated and checked. Exclusive accesses to Shared memory are checked using the internal and external monitor that uses the external memory interface Main interface or P-AHB.

The following table summarizes the processor memory types and associated behavior for data accesses.

**Table 10-1: Memory types and associated behavior for data accesses**

Memory type	Device memory attributes	Shareability	Cacheability	Restartable	Exclusives handled
Normal	-	Shared	No Cacheability	Yes	Internal and external
	-	Non-shared	Only if memory attributes are Cacheable and the cache is present, enabled, and active <sup>1</sup> .	Yes	Internal only
Device	Gathering, G and non-Gathering, nG	Yes	No	No	Internal and external
	Reordering, R and Non-Reordering, nR	Yes	No	No	
	Early Write Acknowledgment, E and No Early Write Acknowledgment, nE	Yes	No	No	



Note

- The Cortex®-M52 processor can merge accesses to Normal memory, but not to Device memory.
- An external interconnect can merge accesses to Normal memory, but must not merge accesses to Device memory.

<sup>1</sup> For more information on cache activity, see [Accessing the caches](#)



- *M-profile Vector Extension (MVE)* instructions to Device memory might merge multiple accesses from the same micro-operation into one transaction, regardless of whether that memory has the Gathering attribute or not.

### 10.3.1 Speculative accesses

The Cortex®-M52 processor performs Speculative accesses to increase performance. The Arm®v8-M and Arm®v8.1-M architecture permit Speculative accesses. System designers must not assume that the scope of the speculation is fixed or definitively specified.

The following list describes some of the examples where Speculative accesses can occur:

- Speculative instruction fetches can be initiated to any Normal, executable memory address. This can occur regardless of whether the fetched instruction gets executed or, in rare cases, whether the memory address contains any valid program instruction.
- Speculative data reads can be initiated to any Normal, read/write, or read-only memory address. In some rare cases, this can occur regardless of whether there is any instruction that causes the data read.
- Speculative reads that target a TCM region can be initiated on any of the three TCM interfaces, regardless of which TCM interface the memory region is mapped to, or whether that address is mapped to any TCM interface.

However, Speculative accesses do not occur in the following cases:

- Speculative instruction fetches on the Main interface are never made to memory addresses in an Execute Never region.
- Speculative data reads are never made to Device memory addresses.
- Speculative reads are never made on the *Peripheral AHB (P-AHB)* and *External Private Peripheral Bus (EPPB)* interfaces.
- Speculative writes are never made.



Memory regions that are mapped to the TCM are always treated as Normal Memory and therefore are always subject to speculation.

#### 10.3.1.1 Considerations for system design

The system designer must ensure that the system is robust enough to handle Speculative accesses, and all executable and Normal type memory regions are safe to access.

#### Preventing Speculative accesses

Speculative accesses do not cause any processor faults. The processor is aware whether an access is Speculative, and ignores any error response that the system signals because of the Speculative access. However, the system in which the processor is integrated in cannot

distinguish between Speculative accesses and Non-speculative accesses. Therefore, the system designer is required to ensure that the system is robust enough to handle Speculative accesses, regardless of whether they are initiated to unexpected memory addresses.

Alternatively, if there are memory regions that are not mapped to the TCMs and to which Speculative access should not be initiated, Arm® recommends setting those regions to have the following attributes with the *Memory Protection Unit* (MPU):

- Device
- Execute-never

In the Cortex®-M52 processor, the following conditions apply for speculative accesses:

- Speculation is not allowed for any access on Main interface for Secure attributed memory regions without Secure access rights.
- Instruction fetches can be made speculatively on Main interface to Normal memory that is not marked as execute-never.
- Data accesses to Normal memory can be speculative on Main interface. In this case, speculative covers data accessed as part of the cache line beyond the specific locations accessed by the instruction.



There are no linefills initiated by an instruction that is not committed in the processor pipeline.

- 
- Data accesses cannot be speculative on P-AHB or EPPB.
  - Instruction fetches are not supported on P-AHB or EPPB.
  - No external bus (Main interface, P-AHB, EPPB) access are made for accesses encountering MPU, SAU or IDAU faults.

The TCMs are always treated as Normal memory. Therefore, they are always subject to speculation.

### MPU, SAU, or IDAU violation behavior

On the Main interface, P-AHB, or EPPB interfaces, an MPU, SAU, or IDAU violation is guaranteed to cause a fault and the access is not initiated on the interface. On the TCM interface, an MPU, SAU, or IDAU violation is guaranteed to cause a fault. However, a read access is still initiated, and in this case, the processor ignores the read data that is returned from the TCM.

## 10.3.2 Access privilege level for Device and Normal memory

The AMBA® 5 AXI, AMBA 5 AHB, and AMBA 4 APB protocols all include signals which allow the privilege level of an access to be reported to the system. The Cortex®-M52 processor supports

these signals across the *AXI Main* (M-AXI), *AHB Main* (M-AHB), *Peripheral AHB* (P-AHB), and *External Private Peripheral Bus* (EPPB) interfaces for Device memory.

The Cortex®-M52 processor also supports privilege reporting for Normal memory on P-AHB. However, Main interface accesses to Normal memory can be buffered and cached so memory read and write requests and instruction fetches from both privileged and unprivileged software can be merged. All Main interface accesses to Normal memory are marked as privileged. For all M-AXI transactions, the AXI signals ARPROT[0] and AWPROT[0] are always 1 indicating a privileged access. For all M-AHB transactions, the AHB signals HPROTC[1] and HPROTS[1] are always 1 indicating a privileged access. Access permission to a region of memory can always be restricted to software running in privileged mode by using the *Memory Protection Unit* (MPU).

The following table shows the processor mode and privilege level values of the read channel protection signal. The security attributes of the transaction are stored in bit 1 of the ARPROT and AWPROT signal.

**Table 10-2: Cortex®-M52 processor mode and read and write channel protection signal privilege information**

Processor mode	Memory type	Value
-	Normal Cacheable	Always marked as Privileged
-	Normal Non-cacheable	
Unprivileged	Device	Unprivileged
Privileged		Privileged

The instruction and data TCM interfaces provide signals ITCMPRIV and D\*TCMPRIV to indicate the privilege of all memory accesses.

For more information on how security attributes are generated and determined, see [Memory Authentication](#).

## 10.4 M-AXI interface

The AXI Main (M-AXI) interface is a single 32-bit AMBA® 5 AXI interface for on-chip or off-chip memory and devices. The interface serves the memory regions that the TCM, *Peripheral AHB* (P-AHB), *Internal Private Peripheral Bus* (IPPB), and *External Private Peripheral Bus* (EPPB) interfaces do not cover.

The M-AXI interface can have either of the following configurations:

- High performance configuration.
- Area optimized configuration.

Both M-AXI configurations provide a store-buffer that supports data merging, reordering, and forwarding for Normal memory to minimize the number of AXI write transactions that are sent out to the system.



- Implementing the L1 data cache results in the high-performance M-AXI configuration. When the L1 data cache is not present, the M-AXI defaults to the area optimized configuration.
- For more information on restrictions and how to provoke the maximum number of outstanding AXI transactions in high performance and area optimized configurations, see [Restrictions on AXI transfers](#).

## 10.4.1 High performance M-AXI configuration

The high performance M-AXI configuration supports extensive buffering and multiple outstanding AXI transactions to optimize memory system performance, even in the presence of large latencies.

This configuration includes a 4-way set associative L1 data cache that supports:

- Read-allocation.
- Write-allocation.
- Write-Back.
- Write-Through.
- Transient.

### 10.4.1.1 High performance configuration M-AXI attributes and transactions

The high performance configuration is designed to be used with a native AXI system with high memory bandwidth and support for multiple outstanding transactions. The following table shows the AXI attributes and transactions that the high performance M-AXI configuration supports.

**Table 10-3: High performance configuration M-AXI attributes and transactions**

AXI attribute	Value	Details
Write issuing capability	31	<ul style="list-style-type: none"> <li>• 15 writes to Device memory.</li> <li>• 16 writes to Normal memory, that can be evictions, write bursts, or single writes.</li> </ul>
Read issuing capability	3	<ul style="list-style-type: none"> <li>• 1 data linefill.</li> <li>• 1 non-cacheable data read.</li> <li>• 1 instruction fetch or instruction linefill for instruction cache (or unified cache) or 1 data read for unified cache.</li> </ul>
Write ID capability	4	<ul style="list-style-type: none"> <li>• 1 reserved for Device memory.</li> <li>• 1 reserved for Normal Non-cacheable writes and exclusive writes.</li> <li>• 1 reserved for Normal cacheable writes</li> <li>• 1 reserved for cache line evictions.</li> </ul>

AXI attribute	Value	Details
Read ID capability	3	<ul style="list-style-type: none"> <li>1 reserved for Normal Non-cacheable and Device memory.</li> <li>1 reserved for Data cache linefill.</li> <li>1 reserved for instruction fetch or instruction linefills.</li> <li>1 reserved for Unified cache linefills.</li> </ul>
Combined issuing capability	34	<ul style="list-style-type: none"> <li>31 outstanding writes.</li> <li>3 reads from data linefills, non-cacheable reads, and instructions fetches.</li> </ul>

Only a subset of all possible AXI transactions can be generated. These are:

- For Normal, Cacheable memory:
  - WRAP8 32-bit reads, for load, store linefills, and instruction linefills.
  - INCR8 32-bit writes, for evictions.
  - INCR N 32-bit or smaller writes with N=1-8 for combined individual no-write allocate stores or if in no Write-Allocate mode.
  - INCR N 32-bit reads with N=1-8, for instruction fetches when the L1 instruction cache is disabled.
- For Normal, Non-cacheable memory:
  - INCR N 32-bit reads with N=1-8 for load multiplies and vector loads.
  - INCR N 32-bit writes with N=1-8 for combined individual stores and store multiples.
  - INCR N 32-bit reads with N=1-8 for instruction fetches.
  - INCR 1 reads of any size, for individual loads.
- For Device memory:
  - INCR 1 32-bit reads for individual load and load multiples.
  - INCR 1 32-bit writes for individual store and store multiples.
  - INCR 1 8-bit, 16-bit reads and writes for individual subword loads and stores.
- INCR 1 8-bit, 16-bit, and 32-bit exclusive reads and writes for shared exclusives.
- No FIXED bursts are used.
- Write bursts to Normal memory can use the following optimizations that are allowed on M-AXI but have implications for bridging to AHB.
  - Entire beats with no strobes set.
  - Non-contiguous strobes per beat.



- INCR is an incrementing burst, where the address for each transfer in the burst is an increment of the address for the previous transfer.
- WRAP is a wrapping burst that is similar to an incrementing burst, except the address wraps around to a lower address if an upper address limit is reached.

- FIXED bursts, which are not used, have the same address for every transfer in the burst.
- For more information on burst types, see the *AMBA® AXI and ACE Protocol Specification*.

## 10.4.2 Area optimized M-AXI configuration

The area optimized M-AXI configuration supports reduced buffering and minimizes the number of outstanding AXI transactions to support a low-cost memory system without the significant area impact of a L1 data cache.

The performance for this configuration is expected to be significantly lower than the configuration described in [High performance M-AXI configuration](#), and this configuration is optimized for area alone, where practical.

### 10.4.2.1 Area optimized configuration M-AXI attributes and transactions

The area optimized configuration is intended to be integrated into a low-cost AXI system or bridged to AHB and is suitable for connection to a low-bandwidth memory system. For example, off-chip memory. The following table shows the AXI attributes and transactions that the area optimized M-AXI configuration supports.

**Table 10-4: Area optimized configuration M-AXI attributes and transactions**

AXI attribute	Value	Details
Write issuing capability	31	<ul style="list-style-type: none"> <li>• 15 writes to Device memory.</li> <li>• 16 writes to Normal memory.</li> </ul>
Read issuing capability	2	<ul style="list-style-type: none"> <li>• 1 data reads.</li> <li>• 1 instruction fetch, or instruction linefill, or data load for unified cache.</li> </ul>
Write ID capability	3	<ul style="list-style-type: none"> <li>• 1 reserved for Device memory.</li> <li>• 1 reserved for Normal memory Non-cacheable writes and exclusive writes.</li> <li>• 1 reserved for Normal cacheable writes.</li> </ul>
Read ID capability	3	<ul style="list-style-type: none"> <li>• 1 reserved for Normal Non-cacheable and Device memory.</li> <li>• 1 reserved for Data cache line-fills.</li> <li>• 1 reserved for instruction fetch or instruction linefills.</li> <li>• 1 reserved for Unified cache linefills.</li> </ul>
Combined issuing capability	33	<ul style="list-style-type: none"> <li>• 31 outstanding writes.</li> <li>• 2 reads from data and instructions fetches.</li> </ul>

Only a subset of all possible AXI transactions can be generated. These are:

- For Normal memory:

- WRAP8 32-bit reads, for instruction linefills, if a L1 instruction cache or unified cache is included.
- INCR N 32-bit reads with N=1-8 for individual loads and load multiples.
- INCR 1 8-bit, 16-bit, and 32-bit reads for individual loads.
- INCR N 32-bit writes with N=1-8 for combined individual stores and store multiples.
- INCR N 32-bit reads with N=1-8, for Non-cacheable instruction fetches or all instruction fetches with no L1 instruction cache.
- For Device memory:
  - INCR 1 32-bit reads for load double or load multiple instructions.
  - INCR 1 32-bit writes for store double or store multiple instructions.
  - INCR 1 8-bit, 16-bit, and 32-bit reads for individual loads.
  - INCR 1 8-bit, 16-bit, and 32-bit writes for individual stores.
- INCR 1 8-bit, 16-bit, and 32-bit exclusive reads and writes for shared exclusives.
- No FIXED bursts are used.
- Write bursts to Normal memory can use the following optimizations that are allowed on AXI but have implications for bridging to AHB.
  - Entire beats with no strobes set.
  - Non-contiguous strobes per beat.



Note

- INCR is an incrementing burst, where the address for each transfer in the burst is an increment of the address for the previous transfer.
- WRAP is a wrapping burst that is similar to an incrementing burst, except the address wraps around to a lower address if an upper address limit is reached.
- FIXED bursts, which are not used, have the same address for every transfer in the burst.
- For more information on burst types, see the *AMBA® AXI and ACE Protocol Specification*.

### 10.4.3 Bridging to AHB

The high performance *AXI Main* (M-AXI) configuration is optimized for a native AXI system and not for AHB. The AHB protocol only allows one outstanding transaction. Therefore, this implies serialization of all outstanding transactions that the M-AXI can support. For acceptable levels of performance, Arm® recommends that at least two AHB interfaces are used in this configuration, one for instructions and one for data.

The area optimized M-AXI configuration can be bridged to a single AHB interface if the resulting performance is acceptable.

Both M-AXI configurations support the following features that need special consideration when bridging to AHB:

#### **Sparse write strobes**

AHB does not support write strobes and therefore must split AXI beats with sparse write strobes into smaller AHB transactions. This implies that AHB write bursts can be used only when the bridge is capable of buffering an entire AXI burst and evaluating the strobes before deciding how to perform the AHB access.

To avoid this issue, the processor provides a sparse write strobe signal. Transactions can use this signal to allow AXI bursts that do not use sparse strobes to be identified before all the write data is provided. Therefore, these accesses can be performed as AHB bursts efficiently. This signal is guaranteed to be valid, but in some cases it might be asserted for transactions that do not have sparse strobes.

#### **Exclusive accesses**

AMBA® AHB protocols prior to AMBA 5 AHB do not support exclusive accesses. Arm recommends all AHB infrastructure used with the Cortex®-M52 processor is based on AMBA® 5 AHB.

The Arm® CoreLink™ AXI5 to AHB5 XHB-500 bridge, which is included in the Arm® Corstone-300 Foundation IP, can be used with Cortex®-M52, and also supports the sparse write strobes signal.

### **10.4.4 Write response**

It is a requirement of the systems using the AMBA® 5 AXI protocol that the slave does not return a write response until it has received the write address.

### **10.4.5 Memory system implications for AXI accesses**

The attributes of the memory being accessed can affect an AXI access.

The memory system can cache any cacheable Normal memory address that has either the Read-Allocate or Write-Allocate hint.

Accesses to Device memory cannot be cached and are always Outer Shareable. Any unaligned access to device memory generates an UNALIGNED UsageFault exception and therefore does not cause an AXI transfer.

Normal Non-cacheable memory can also be Outer Shareable.



Memory regions marked as Non-Cacheable Normal must not be used to access read-sensitive peripherals in a system. This is because read transactions to these regions from the processor can be repeated multiple times if the originating load instruction is interrupted.

---



## 10.4.6 M-AXI interface transfers

The AXI Main (M-AXI) interface does not generate the following types of transactions:

- An AXI slave device connected to the M-AXI interface must be capable of handling every kind of transaction that the *AMBA® AXI and ACE Protocol Specification* permits, except where there is an explicit statement in this chapter that such a transaction is not generated. You must not infer any additional restrictions from the example tables given.
- Non-cacheable load instructions might not result in an AXI transfer if they forward from an internal buffer.
- Non-cacheable store instructions always result in an AXI transfer, but multiple stores might get merged into one AXI transaction.
- If the processor is powered up, the buffered write response ready signals, BREADY is always asserted. You must not make any other assumptions about the AXI handshaking signals, except that they conform to the *AMBA® AXI and ACE Protocol Specification*.

### 10.4.6.1 Restrictions on AXI transfers

The AXI Main (M-AXI) interface applies restrictions to the AXI transactions it generates.

These restrictions are:

- A burst never transfers more than 32 bytes.
- The burst length is never more than eight transfers.
- The maximum length of a Device write burst is one transfers. Device reads are always a single transfer.
- No transaction ever crosses a 32-byte boundary in memory.
- FIXED bursts are never used.
- The write address channel always issues INCR type bursts, and never WRAP or FIXED.
- If the transfer size is 8 or 16 bits then the burst length is always one transfer.
- The transfer size is never greater than 32 bits, because it is a 32-bit AXI bus.
- Instruction fetches are always a 32-bit transfer size, and never locked or exclusive.
- Exclusive accesses are always to addresses that are aligned for the transfer size.
- Only exclusive accesses to shared memory result in exclusive accesses on the M-AXI. Exclusive accesses to non-shared memory are marked as non-exclusive accesses on the bus.
- For high-performance M-AXI configurations, to observe the maximum number of outstanding accesses, the M-AXI interface must be very slow so that the following sequence can be performed before any write response and read data response for an access in the sequence occurs:
  1. Execute a `DSB` instruction.
  2. Perform one `PLD` instructions to Read-Allocate Cacheable memory. The cache line must not already in the cache.

3. Perform one of the following actions:
  - Trigger 7 evictions through cache maintenance operations. This requires prior allocation of 7 cache lines into the data cache and making these cache lines dirty with store transactions.
  - Execute 7 byte stores to Cacheable, No-write Allocate memory. Each store must be to a separate cache line.
4. Execute 15 byte stores to Device memory.
5. Execute 9 byte stores to Non-cacheable memory. Each store must be to a separate cache line.
6. Trigger an instruction side fetch from an address that is Cacheable and not already in the instruction cache.
7. Execute a `DSB` instruction.



Note

This sequence can only achieve 2 outstanding read and 31 outstanding write accesses at the same time. Simultaneous 3 reads and 31 writes can only be observed when there happens to exist an individual load transaction from Non-cacheable memory before evictions or stores mentioned above, and that individual load transaction is killed in pipeline after it is initiated on AXI bus.

- For area-optimized M-AXI configurations, to observe the maximum number of outstanding accesses, the M-AXI interface must be very slow so that the following sequence can be performed before any write response and read data response for an access in the sequence occurs:
  1. Execute a `DSB` instruction.
  2. Execute 7 byte stores to cacheable memory. Each store must be to a separate cache line.
  3. Execute 15 byte stores to Device memory.
  4. Execute 9 byte stores to Non-cacheable memory. Each store must be to a separate cache line.
  5. Trigger an instruction side fetch from an address that is Cacheable and not already in the instruction cache or unified cache.
  6. Execute a `DSB` instruction.



Note

This sequence can only achieve 1 outstanding read and 31 outstanding write accesses at the same time. Simultaneous 2 reads and 31 writes can only be observed when there happens to exist an individual load transaction from Non-cacheable memory before stores mentioned above, and that individual load transaction is killed in pipeline after it is initiated on AXI bus.

## 10.5 AHB Main (M-AHB) interface

The AHB Main (M-AHB) interface is two single 32-bit AMBA® 5 AHB interfaces for on-chip or off-chip memory and devices. The interfaces serve the memory regions that the TCM, *Peripheral AHB* (P-AHB), *Internal Private Peripheral Bus* (IPPB), and *External Private Peripheral Bus* (EPPB) interfaces do not cover.

The M-AHB interface is comprised of two AHB interfaces:

- Code main interface (CODE-AHB) is mapped to code region that is not covered by ITCM.
- System main interface (SYS-AHB) is mapped to other regions that are not covered by DTCM, P-AHB, or PPB.

Cortex®-M52 can send one request to CODE-AHB and one to SYS-AHB in parallel.

M-AHB configurations provide a similar store-buffer that supports data merging, reordering, and forwarding for Normal memory to minimize the number of AHB write transactions that are sent out to the system. When perform M-AHB configurations, pay attention to the following items:

- L1 cache configuration has impact in M-AHB.
  - When data cache is included, cache eviction is supported by M-AHB.
  - When data cache is not included, no cache eviction is not supported by M-AHB.
- M-AHB does not support burst write initiated by stores from store-buffer.
- M-AHB can live with an L1 unified cache.

Only the following AHB transactions are supported:

- For Normal, Cacheable memory:
  - WRAP8 32-bit reads (for load and store linefills and instruction linefills)
  - INCR8 32-bit writes (for evictions when data cache is configured)
  - SINGLE 8-bit, 16-bit and 32-bit writes (for coalesced individual no-write allocate stores, or in no write allocate mode)
  - INCR 32-bit reads with undefined length (for instruction fetches when the Instruction cache or unified cache included in your system is disabled)
  - SINGLE 8-bit, 16-bit, and 32-bit reads (for individual loads)
- For Normal, Non-cacheable memory:
  - INCR 32-bit reads with undefined length (for load multiplies and vector loads)
  - SINGLE 8-bit, 16-bit and 32-bit reads (for individual loads)
  - SINGLE 8-bit, 16-bit, and 32-bit writes (for coalesced individual stores and store multiples)
  - INCR 32-bit reads with undefined length (for instruction fetches)
- For Device memory:
  - SINGLE 32-bit reads (for individual loads, load multiples, and load doubles)
  - SINGLE 32-bit writes (for individual stores, store multiples, and store doubles)

- SINGLE 8-bit and 16-bit reads and writes (for individual sub-word loads and stores)
- SINGLE 8-bit, 16-bit, and 32-bit exclusive reads and writes (for shared exclusives)
- No WRAP4/INCR4/WRAP16/INCR16 bursts are used

## 10.6 Peripheral AHB interface

The *Peripheral AHB* (P-AHB) interface is a single 32-bit wide interface that conforms to the AMBA® 5 AHB protocol. It is designed for deterministic, data-only access to fast on-chip peripherals.

### 10.6.1 P-AHB interface transfers

For each clock cycle, the *Peripheral AHB* (P-AHB) interface supports one aligned 32-bit access or any 8-bit or 16-bit access that can fit inside an aligned 32-bit access. Unaligned accesses that cross a 32-bit boundary are split into multiple accesses.

#### Memory region type

By default, the memory regions mapped to the P-AHB interface are Device, however, it is possible to map regions as Normal using the *Memory Protection Unit* (MPU). Although Normal memory is supported on the P-AHB interface, Normal memory-specific optimization is not allowed. This implies that the interface is generally unsuitable for high-bandwidth requirements, and for such a requirement, the *Tightly Coupled Memory* (TCM) or Main interface must be used instead.

#### Unaligned request support

The P-AHB can accommodate unaligned requests to Normal memory by breaking down the request into a set of aligned transactions that is suitable for its protocol. In most cases, the number of accesses to complete an unaligned write is greater than an equivalent read because if required, Normal memory can be excessively read, but the P-AHB interface does not support partial writes. [Unaligned memory access timing](#) lists the number of individual read and write transactions that are generated for the unaligned transactions.

#### Instruction execution and vector fetches support

Instruction execution and vector fetches are not supported on this interface. The P-AHB is targeted at on-chip peripherals only. Instruction and vector fetches to P-AHB are sent on the M-AXI/M-AHB interface.

#### Transactions supported

New transactions cannot be started on the bus until all outstanding transactions are completed. This implies all transactions to this interface are in-order. Loads can only start on the bus after all buffered writes are drained.

The P-AHB does not support burst transactions. This implies that, the P-AHB interface only uses one transfer and all bursts are single.

The P-AHB does not support Speculative accesses, write merging, and forwarding of buffered store data for reads. No transaction ever crosses a 4-byte boundary in memory. The transfer type is never SEQUENTIAL.

Exclusive accesses are supported in the P-AHB interface, and these accesses are always to addresses that are aligned for the transfer size. Exclusive transactions are only generated for Shareable memory regions.

The P-AHB interface can also break down sparse reads and writes that are associated with the *M-profile Vector Extension* (MVE) Load and Store instructions.

Multiple write transactions can be buffered more than once, therefore, more than one imprecise BusFault exception can be raised because of external errors. The exceptions are always raised in the same order of the store instructions which generated the transactions.

The following table assumes that only non-MVE write accesses are considered. For unaligned MVE writes, the number of accesses changes depending on the element size and predicate mask. For more information, see the *Arm®v8-M Architecture Reference Manual*.

**Table 10-5: Unaligned memory access timing**

Access size	Address offset	Number of read accesses	Number of write accesses
Word	+1	2	3
	+2	2	2
	+3	2	3
Halfword	+1	1	2
	+3	2	2



Arm® recommends that the P-AHB is reserved for low-latency peripherals and all others are integrated on the Main interface. This allows:

- Better overall processor execution performance in the presence of frequent stores to high-latency peripherals.
- Better *Quality of Service* (QoS) to P-AHB peripherals in interrupt handlers that do not make frequent accesses to high-latency peripherals on the Main interface.

## 10.6.2 P-AHB interface configuration

The *Peripheral AHB* (P-AHB) interface covers two ranges in the processor memory map, that is, the Peripheral region and the Vendor\_SYS region.

### Peripheral region

Base address is fixed at 0x40000000. The P-AHB region starts at the base address and has a size determined by PAHBCR.SZ, which is configured using the input signal CFGPAHBSZ.

### Vendor\_SYS region

The address range is 0xE0100000-0xFFFFFFFF.

Mapping the Vendor\_SYS region of the memory map to the P-AHB interface allows existing AHB-based peripherals designed for M-profile systems to be reused in Cortex®-M52-based designs.

Mapping the Vendor\_SYS region of the memory map to the P-AHB interface provides additional, always-enabled, address space for direct connection to AHB-based slaves, for example, re-used peripherals from existing Cortex-M systems.

The following parameters can be controlled for the P-AHB:

#### Size

The external input signal CFGPAHBSZ controls the size of the Peripheral region mapped to the P-AHB interface. This signal can only be changed at Cold reset. A maximum of 0.5GB is supported. This implies that the P-AHB interface is present entirely in the Peripheral region and can cover it completely. The Vendor\_SYS region size is not configurable.

#### Enable

The external input signal INITPAHBEN controls the P-AHB enable state at reset. During runtime, the P-AHB Peripheral region can be enabled and disabled using the PAHBSCR register. Only privileged software can modify this register.

Also, if the Security Extension is implemented and AIRCR.BFHFNMINS is zero, this register is RAZ/WI from Non-secure state. The Vendor\_SYS region is always enabled.

#### Alias

The P-AHB interface supports the ability to alias two logical addresses in the Peripheral region onto the P-AHB interface. This feature is used with an external security gate to support fine-grain Secure and Non-secure regions in the Peripheral region. The alias bit in the logical address can be configured from bit[26] to bit[28] using the external input signal CFGMEMALIAS. This signal can only be changed at reset.

Data accesses to the P-AHB Peripheral region are performed on the Main interface when the P-AHB interface is disabled. Accesses to the Peripheral region above the P-AHB size limit are also performed on the Main interface.

Instruction accesses made to the Peripheral region, where executable, are always performed on the M-AXI interface. For code portability, Arm® recommends that the P-AHB region is programmed as *Execute Never* (XN) in the *Memory Protection Unit* (MPU) to prevent instruction execution. This is consistent with the default memory map. The Vendor\_SYS region is permanently XN.

### 10.6.3 P-AHB considerations

Normal memory is supported on the *Peripheral AHB* (P-AHB) interface. However, no Normal-specific optimizations are made. This means the interface is generally not suitable for high-bandwidth requirements, and the *Tightly Coupled Memory* (TCM) or Main interface must be used instead.

Instruction execution and vector fetches are not supported on this interface. The P-AHB is targeted at on-chip peripherals only.

The amount of buffering resource is intentionally limited to provide a balance between load access latency and store throughput. The implications of this limited buffering are:

- Individual stores to the P-AHB interface are visible to the Device memory in minimal and deterministic time relative to the store instruction being executed. This is relevant, for example, when an interrupt handler must perform a critical device access.
- There is limited hiding of store latency from the pipeline. This means that high-latency peripherals can stall the pipeline on a store instruction for extended periods of time. However, it affects the overall processor execution performance.
- Loads to the P-AHB interface are inherently higher latency than stores and must wait for all buffered stores to drain before they can be started on the bus. The limited buffering means that this latency is minimized but can still be significant for high-latency peripherals. The pipeline cannot flush a load that has started on the bus. Therefore, interrupt latency is affected by wait-states on loads. However, loads that have not yet started on the bus can be safely flushed. Therefore, the impact of load wait-states on interrupt latency is limited to the wait-states on a single access.
- Load access throughput is limited. There is no support for bursts on load multiples and no support for pipelined loads in general.
- Store throughput is acceptable for zero wait state systems, but it is degraded when wait states are used.

## 10.7 TCM-AHB interface

The 32-bit *AHB TCM Access* (TCM-AHB) interface provides system access to the *Tightly Coupled Memories* (TCMs). Typically, a *Direct Memory Access* (DMA) controller uses this interface to transfer data in and out of the processor for software computation. It includes arbitration logic to support simultaneous system and processor TCM access requests. The TCM-AHB interface implements the AMBA® 5 AHB protocol.

If there is no contention with software access to TCM and the TCM uses zero wait states, then write buffering and the read prefetcher allows the TCM-AHB interface to indefinitely sustain back-to-back write and read transactions.

### Write buffering

Writes are buffered in the TCM-AHB interface to improve system performance.

Read access latency is inherently larger than write access latency because the AHB interface can only support a single outstanding transaction. To minimize this latency, reads can overtake buffered writes. However, if there is a data dependency between a read and a buffered write, then hazarding logic stalls the read and attempts to drain the buffer until there are no longer any dependencies. Writes are always carried out in-order and hazarding is performed at byte granularity.

Additional hazarding is included to fully serialize read accesses to the TCM from the TCM-AHB interface and software running on the processor. This allows both masters to access the TCM coherently. For more information on how data can be shared between software running on the processor and system-level devices that are connected on the TCM-AHB interface, see [System access to TCM through the TCM-AHB DMA interface](#).

## Read prefetcher

The TCM-AHB interface also supports a read prefetcher to improve the performance of reading bursts of data from the TCM to the system. The prefetcher supports the following 32-bit read transfers:

- INCR.
- INCR4.
- INCR8.
- INCR16.

If there is no contention or wait states on the TCM banks being accessed, the prefetcher generates internal transactions so that read data can be returned on consecutive clock cycles on the TCM-AHB interface.



Note

- The TCM-AHB interface supports an extension to the AHB5 protocol using byte-lane strobe signals to efficiently handle data with non-contiguous write-data in a beat, similar to that supported on AMBA AXI interfaces. This allows for efficient bridging from an AXI-based DMA controller.
- All TCM-AHB accesses are treated as being the same endianness as memory. No data swizzling is performed for reads or writes.
- The TCM-AHB interface can be used even if the processor is in sleep mode.

### 10.7.1 TCM-AHB memory map

The memory map that is presented on the *AHB TCM Access* (TCM-AHB) interface is consistent with the memory map that is presented to software running on the processor. Only the *Tightly Coupled Memory* (TCM) address range can be accessed. Any other addresses cause an AHB fault response.

The following table shows the TCM-AHB memory map. The TCM Start address is configurable. See [TCM configuration](#) for more information about the Configurable TCM Start address.

**Table 10-6: TCM-AHB memory map**

Start address	End address	Bits [2] on the system address bus, HADDRS[2]	TCM accesses	TCM index
0x00000000	0x00000000+ ITCM size	-	ITCM	HADDRS[n:2] <b>Note:</b> The value of n depends on the configured TCM size.
0x20000000	0x20000000+ DTCM size	0	D0TCM	HADDRS[n:3]
0x20000000	0x20000000+ DTCM size	1	D1TCM	HADDRS[n:3]



A read or write request on the TCM-AHB interface to the SRAM region is mapped to 32-bit accesses to one separate DTCM instance according to HADDRS[2].



- The TCM enable fields that are defined in the TCM control registers, ITCMCR and DTCMCR, do not affect TCM-AHB accesses.
- If Security gating is enabled on the TCM interface, the address ranges are aliased in the same manner as defined for software access.

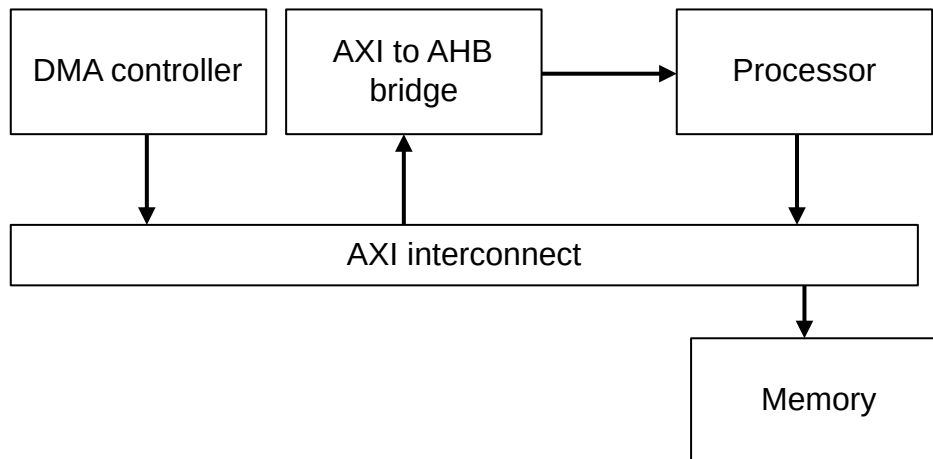
## 10.7.2 TCM-AHB transfers

The *AHB TCM Access* (TCM-AHB) interface has certain conditions that require consideration.

The Cortex®-M52 processor does not support TCM-AHB transactions that are directly dependent on software memory transactions. This means that the system must not introduce any dependencies which imply that a software memory access cannot complete until a corresponding TCM-AHB transaction completes. Therefore, no loopback arrangements from processor master ports to the TCM-AHB interface are supported because these arrangements might cause deadlock. This restriction does not prevent arrangements where software memory-mapped accesses are used, for example, on the *Main bus* (M-AXI/M-AHB) or *Peripheral AHB* (P-AHB) interface, to request an external agent to perform transactions on the TCM-AHB. The only requirement is that there is no dependency introduced in the system between the control access that initiates the transaction and the transaction itself.

If a system integration contains an example design as shown in the following figure, the address decoder in the AXI interconnect that is coupled to the Cortex®-M52 processor must be configured so that the address range for the TCM-AHB interface is blocked to ensure that this requirement is met. The address decoder logic for other masters that are connected to the AXI interconnect are not affected by this requirement.

**Figure 10-1: Example system integration**



TCM-AHB transactions cannot perform *Memory Protection Unit* (MPU) lookups. There is no internal distinction between unprivileged and privileged TCM-AHB accesses. The system is entirely responsible for providing TCM protection functionality for TCM-AHB accesses as required. This can be carried out by performing a privilege check in either of the following areas:

- When the system memory agent has been requested for the access. This is entirely system defined and no specific hardware support is provided.
- When the TCM-AHB access is performed on the TCM interface. In this case, the hardware performs the TCM access at the privilege level of the TCM-AHB request.
- The TCM-AHB does not support exclusive or locked accesses and TCM-AHB writes do not affect the state of the internal exclusive access monitor, making it unsuitable for systems requiring concurrency controls between the TCM-AHB and software.

The security level for TCM-AHB transactions is indicated by the HNONSECS signal on the interface. This signal indicates the fully attributed security level. That is, after any system-level *Implementation Defined Attribution Unit* (IDAU), TCM-AHB accesses are not passed through or checked against the processor IDAU or *Security Attribution Unit* (SAU). The TCM security gate can be used to control access to the TCM based on the transaction security level.



- INCR is an incrementing burst, where the address for each transfer in the burst is an increment of the address for the previous transfer.
- For more information on burst types, see the *Arm® AMBA® 5 AHB Protocol Specification*.

For more information on TCM security gating, see [TCM and P-AHB security access control](#).

### 10.7.3 TCM-AHB interface arbitration

In normal operation, there is enough bandwidth across the two *Data Tightly Coupled Memory* (DTCM) interfaces to allow accesses from software and the *AHB TCM Access* (TCM-AHB) interface to sustain their maximum throughput and the *Instruction Tightly Coupled Memory* (ITCM) is normally only used for instruction fetch. This means contention for resource should be rare and so the TCM-AHB is usually the lowest priority with no impact on the performance of data transfer from the system to the TCM.

However, there might be cases when a source makes large numbers of accesses to the same TCM bank. To prevent the TCM-AHB interface from getting less bandwidth, the priority of a request on the interface is automatically boosted when there is contention with a software access. When this occurs, a round robin scheme is used to share the bandwidth to a TCM bank roughly equally between TCM-AHB accesses and software accesses. This also allows the TCM bandwidth to be split evenly between software and TCM-AHB transactions if contention occurs.

### 10.7.4 TCM-AHB availability and low power states

The following conditions are required for the TCM-AHB to accept transactions:

- The processor power domain (PDCORE) is active and not in reset.
- CLKIN is running.

The TCM-AHB sub-system and the TCMs are in a separate internal clock domain to the rest of the processor. However, they are in the same reset and power domains. Therefore, TCM-AHB transactions can be performed without the main internal processor clock running. This allows TCM data transfers to be offloaded to a low-power system agent while the processor is in any of its sleep modes. The TCM clock is gated inside the processor to minimize the power used when no transactions are in progress from either the processor or TCM-AHB. Asserting HTRANSS automatically starts the clock if it is gated and the clock is stopped after all outstanding transactions have completed. For more information on HTRANSS, see [TCM-AHB interface signals](#)



From a system perspective, you are responsible to ensure that CLKIN is running when a transaction is started on TCM-AHB by considering the requirements of any master components which can access the slave interface, for example a DMA, and enabling system level clock gating accordingly. This might mean overriding the current CLKINQACTIVE state if the processor is in sleep and so not requesting CLKIN.

---

## 10.8 EPPB interface

The *External Private Peripheral Bus* (EPPB) interface is a 32-bit AMBA® 4 APB interface designed for integration with CoreSight™ debug and trace components.

It is used for data accesses to the memory region 0xE0040000-0xE00FFFFF. Instruction accesses to this region cause a fault, and are permanently disabled in the Arm®v8.1-M architecture.

The interface is not intended for general peripheral usage and has both higher latency and lower average throughput than the Main interface or *Peripheral AHB* (P-AHB) interfaces. Additionally, it has the following limitations that make it unsuitable for general-purpose use:

- Only little-endian accesses are supported. This indicates that the processor endianness is ignored.
- All accesses are treated as Device transactions.
- Only aligned accesses are supported. Unaligned accesses to the EPPB interface cause an UNALIGNED UsageFault.
- Exclusive accesses are not supported.
- Only Privileged accesses are supported. Unprivileged accesses take a BusFault exception.

Arm® recommends that all non-debug peripherals are integrated on the Main interface or P-AHB interface.

The EPPB interface can perform debugger-initiated transactions during processor reset. The EPPB interface can also be extended to support interface protection between the processor and the interconnect. For more information on interface protection, see [Interface protection behavior](#).

For more debugging information, see [Debug during reset and before code execution commences](#).

Additionally, for more information on EPPB peripherals, see [Private Peripheral Bus](#).

The EPPB interface is also used to transfer *Nested Vectored Interrupt Controller* (NVIC) state to an *External Wakeup Interrupt Controller* (EWIC) on sleep entry and exit. For more information on EWIC sleep entry and exit, see the *Arm® Cortex®-M52 Processor Integration and Implementation Manual*.



The *Arm® Cortex®-M52 Processor Integration and Implementation Manual* is a confidential document that is only available to Cortex®-M52 processor IP licensees and Arm® partners with an NDA agreement.

---

## 10.9 TCM interfaces

The *Tightly Coupled Memory* (TCM) interfaces are tightly coupled into the processor for optimum performance from fast on-chip memory.

The Cortex®-M52 processor supports two separate interface groups:

### ITCM

Single 32-bit interface that is intended for instruction memory based on SRAM or potentially flash memory with system prefetch or acceleration.

### DTCM

Two 32-bit interfaces intended for use with data memory that is expected to be based on SRAM. The Cortex®-M52 processor performs address filtering that is based on bits[2] of the address.

- Addresses with bit[2]=0b0 are performed on the D0TCM interface.
- Addresses with bit[2]=0b1 are performed on the D1TCM interface.

This configuration requires that the DTCM RAM is logically arranged into two separate address banks. This allows:

- Up to 64 bits of total bandwidth for software reads and writes, and *Direct Memory Access* (DMA) traffic through the *AHB TCM Access* (TCM-AHB) interface with a probabilistic reduction of contention. This is essential for compute performance because the Cortex®-M52 processor can sustain a data throughput of 32 bits per cycle using the *M-class Vector Extension* (MVE) instructions.
- A 32-bit bandwidth for contiguous accesses that are 32-bit aligned from the software and DMA. A 32-bit bandwidth for contiguous accesses is essential for both overall performance and interrupt latency.



Note

- 
- The Cortex®-M52 processor does not provide software control over address filtering.
  - All TCM interfaces support wait and error response from external memory. For systems where functional safety or *Reliability, Availability, and Serviceability* (RAS) are required, the Cortex®-M52 processor also optionally supports a *Single Error Correction and Double Error Detection* (SECCDED) scheme that is based on the *Error Correcting Code* (ECC) for all accesses in the ITCM and DTCM regions.
  - To configure the processor to support ECC, see the configuration options in the *Arm China Cortex®-M52 Processor Integration and Implementation Manual*. The *Arm China Cortex®-M52 Processor Integration and Implementation Manual* is only available to licensees.
-

## 10.9.1 TCM configuration

The TCM interface has fixed and configurable parameters.

The default address of each TCM is fixed:

### ITCM

0x00000000. This is the base address of the Code region.

### DTCM

0x20000000. This is the base address of the SRAM region.

The following parameters can be separately controlled for each of the TCMs:

<b>Size</b>	External configuration input signals control the size of each TCM region. These signals can only be changed at Cold reset. A maximum of 16MB for each TCM is supported. This implies that the ITCM and DTCM are present entirely in the Code and SRAM regions of the memory map respectively.
<b>Enable</b>	An external input signal controls the TCM enable state at reset. During runtime the TCM can be enabled and disabled using the ITCMCR and DTCMCR registers. Only privileged software can modify these registers. If the Security Extension is implemented and AIRCR.BFHFNMINS is zero, these registers are RAZ/WI from Non-secure state.
<b>Alias</b>	The TCM controller can alias two logical addresses in the Code and SRAM regions onto the ITCM and DTCM interface respectively. This feature is used with the TCM security gate to support fine-grain Secure and Non-secure regions in TCM memory. The alias bit in the logical address can be configured from bit[26] to bit[28] using the external input signal CFGMEMALIAS. This signal can only be changed at Cold reset.
<b>Relocate</b>	Base Address of ITCM and DTCM can be relocated. By default, the base address of ITCM and DTCM is fixed at 00000000 and 20000000. To support flex application with multiple Cortex®-M52 processors, Cortex®-M52 supports relocatable base address of ITCM and DTCM. The relevant configuration input signal is CFGCPUINST [1:0]. If relocatable base address is not needed in a single processor or multiple-processor system, CFGCPUINST [1:0] must be tied to 00 to ensure Cortex®-M52 can work in default way. If relocate the base address is needed, tie the CFGCPUINST[1:0] to the value you intend. The final TCM base address bit[25:24] of each Cortex®-M52 must be equal to CFGCPUINST[1:0] of each Cortex®-M52. For example, if CFGCPUINST[1:0] is tied to 01, the base address of ITCM and DTCM is 01000000 and 21000000. Software must be programmed to use the relocatable address to access ITCM and DTCM, based on CFGCPUINST value. Alias address can be used together with relocatable base address.

For a system with four Cortex®-M52 processors instantiated, the base address of each Cortex®-M52 TCM is listed in the following table.

**Table 10-7: Example of relocated base address of TCM**

Cortex®-M52 instances	ITCM base address	DTCM base address
CPU0	00000000	20000000
CPU1	01000000	21000000
CPU2	02000000	22000000
CPU3	03000000	23000000



- For more information on ITCMCR and DTCMCR registers, see [ITCMCR and DTCMCR, TCM Control Registers](#).
- For more information on AIRCR, see the *Arm®v8-M Architecture Reference Manual*.
- Address aliasing and security gating are described in [TCM and P-AHB security access control](#).
- To configure the processor to support ECC in the TCMs, set `ecc` to TRUE. See the *Arm China Cortex®-M52 Processor Integration and Implementation Manual*. The *Arm China Cortex®-M52 Processor Integration and Implementation Manual* is a confidential document that is available only to licensees and Arm partners with an NDA agreement.

## 10.9.2 TCM transactions

TCM regions are implicitly Normal, Non-shareable, Non-cacheable memory.

For TCM memory regions, the Cortex®-M52 processor:

- Ignores the *Memory Protection Unit* (MPU) memory type attributes that software assigns. The MPU protection settings are always considered.
- Initiates Speculative reads. You must not assume that the scope of this speculation is fixed, or that it can be definitively specified. For example, speculation might occur:
  - For instruction prefetching, depending on the recent execution stream.
  - For data reads that are performed before the Security or MPU protection settings are evaluated. Although the access might be performed speculatively, an abort is subsequently raised if required by the Security or MPU protection settings.
  - For data reads in branch shadows.
- Buffers data on writes. Read transactions always hazard against outstanding buffered write transactions to the same address. Writes transactions are never Speculative.

This behavior makes TCMs unsuitable for peripherals or any memory that has implications for read transactions. Devices of this type must be integrated on the *Peripheral AHB* (P-AHB) or Main interface. These interfaces support the Device memory type. Additionally, the following accesses are performed on the Main interface instead of the TCM interfaces:

- Accesses to TCM regions when the relevant TCM is disabled.

- Accesses to the Code and SRAM regions above the TCM size limit, regardless of the TCM enable.

For code portability to other Arm® processors or systems, Arm® recommends that TCM regions are always defined as Normal, Non-shareable memory in the MPU.

This is consistent with the default memory map attributes which apply when the MPU is either disabled or not implemented.

### 10.9.3 Booting from TCM

The Cortex®-M52 processor provides support for booting from volatile TCM memory that must be initialized at reset.

The TCMs can be enabled out of reset without software programming. When the CPUWAIT signal that stalls the core is HIGH out of reset, it prevents the processor from executing any software at the reset vector. This allows the TCMs to be loaded by the system before the processor performs any TCM accesses. When the TCM loading sequence is complete, this signal can be deasserted to allow the processor to boot up. The *AHB TCM Access (TCM-AHB) Direct Memory Access (DMA)* interface is functional when the CPUWAIT signal that stalls the core is asserted out of reset and can therefore service transactions that the system initiates to load the TCMs. This avoids the need for external hardware on the TCM interface for boot-time initialization.



Note

Asserting CPUWAIT prevents the processor from reading the stack pointer (SP) or initial program counter (PC) from the reset vector. Therefore, it is safe to load the vector table, code, and data into the TCM. Alternatively, the external input signals INITSVTOR and INITNSVTOR can be used to set the vector table address in non-volatile memory.

When ECC is enabled, before performing a byte, halfword, or partial word write to a TCM location which causes an RMW, you must initialize the location first by performing an aligned word write to the location. Arm® recommends that all TCM locations are initialized in this manner by boot code.

### 10.9.4 Integration with flash memory

The Cortex®-M52 processor can support the use of flash memory connected to *Tightly Coupled Memory* (TCM). The *Instruction Tightly Coupled Memory* (ITCM) interface is most suitable for this arrangement.

The system must take into account the fetch bandwidth requirements for efficient code execution by the processor. The processor can consume up to 32 bits of instruction data per cycle using both 32-bit Thumb and 16-bit Thumb instructions, because the 16-bit Thumb instructions can be dual-issued. The overall bandwidth is specific to your application but for general-purpose products, it must be assumed that 32 bits per cycle might be required. The instruction memory system needs to sustain this for maximum performance. Arm® recommends that if flash memory is integrated



on the ITCM, some system cache or Flash accelerator is used to meet these fetch bandwidth requirements.

Alternatively, flash memory can be integrated on the Main interface and the processor can be configured to include an L1 instruction cache or unified cache.

### 10.9.5 System access to TCM through the TCM-AHB DMA interface

The 32-bit *AHB TCM Access* (TCM-AHB) interface provides system access to the *Tightly Coupled Memory* (TCM) even when the Cortex®-M52 processor is running.

Typically, this feature is used with a (DMA) controller to transfer data to and from the processor for compute applications. Arbitration between processor access from software and Direct Memory Access TCM-AHB requests to TCM is fully supported with no requirement for external TCM interface logic. For more information on this arbitration logic, see [TCM-AHB interface arbitration](#).

There is no hardware support for concurrency control between software and TCM-AHB access to TCM. Particularly, software exclusive accesses to TCM are only subject to the internal exclusive monitor which does not take TCM-AHB accesses into consideration. This implies that the system must not perform TCM-AHB accesses to any regions of TCM memory that are used with software exclusive accesses. However, it is possible in software to share data coherently between the executing thread and the TCM-AHB interface. The processor makes the following hardware guarantees to share data coherently:

- Appropriate writes to the TCM by software and TCM-AHB are never repeated. Store double instructions, floating-point store multiple instructions storing double-precision values, *M-profile Vector Extension* (MVE) stores, and unaligned single stores can be repeated on exception return. Therefore, these transactions are exempt from this guarantee and unsuitable for software synchronization. The processor guarantees that no single-copy-atomic access is repeated.
- Software and TCM-AHB writes to the TCM have a single point of serialization which is the *TCM Control Unit* (TCU). This means that when a write is observable by one master, it is guaranteed to be observable by the other.
- When a write on the TCM-AHB interface is accepted, the processor assumes responsibility for the coherent observation of that data. Any read by any master interface that is initiated after the TCM-AHB write completed returns the updated data.



- TCMs are implicitly Normal memory, therefore, write buffering is permitted.
  - All TCU buffers are drained before the processor enters a low-power sleep state.
- 

The following table shows an example software sequence for message passing between coherent components in a system.

**Table 10-8: Example software sequence for message passing between coherent components in a system**

Data generator	Data consumer
STR <data>	LDR <valid>
STL <valid> : Store-release	LOOP Direct Memory until <valid> set
	LDA <data> : Load acquire

The TCM-AHB interface always performs writes in-order, and therefore, it does not need a barrier when generating data into the TCM.

Interrupt-based synchronization is also possible in the Cortex®-M52 processor when the TCM-AHB is the data generator. In this model, an interrupt is generated when the last data transfer completes on the external interface. The first instruction in the *Interrupt Service Routine* (ISR) is guaranteed to observe any data items that are stored before or on this transfer. In this case, the completion of the last TCM-AHB access is used to indicate global observability instead of performing a software read of the location and waiting until it has been updated.

For more information on the TCM-AHB interface, see [TCM-AHB interface](#).

## 10.10 Instruction cache, data cache, and unified cache

The Cortex®-M52 processor supports optional, internal L1 Harvard caches for high performance operation using on-chip or external memory. It also supports an optional, internal L1 unified cache. The unified cache can only be included when both data cache and instruction cache are not been included.

Only the Main interface accesses can be cached. TCM and *Peripheral AHB* (P-AHB) interface transactions or accesses cannot be cached.

To enable software to appropriately deal with different levels of cache, the cache maintenance operations can perform up to the following points:

### **Point of Unification (PoU)**

This is the point at which the instruction cache, data cache, and unified cache can see the same copy of a memory location. For the Cortex®-M52 processor:

- When an L1 data cache or an instruction cache or unified cache is included, the PoU is always at the system level, therefore, cache maintenance operations by address always act on the L1 cache. This is indicated by CLIDR.LoUU and CLIDR.LoUIS bitfields. This implies that the data, instruction and unified cache accesses are unified at the system level.
- When the data cache, instruction cache, and unified cache are excluded, the CLIDR.LoUU and CLIDR.LoUIS bits are 0b000.

### **Point of Coherency (PoC)**

This is the point at which all components that can access memory can see the same copy of a memory location. For the Cortex®-M52 processor:

- When an L1 data cache or instruction cache or unified cache is included, the PoC is always at the system level, therefore, cache maintenance operations by address always

act on the L1 cache. This is indicated by CLIDR.LoC bit field. This implies that data accesses are coherent at the system level or beyond the system level.

- When the L1 data cache and instruction cache and unified cache are excluded, the CLIDR.LoC bit is 0b000.

For more information on the CLIDR register, see [CLIDR, Cache Level ID Register](#).

Each cache can be independently configured within the following range:

- 1KB (only unified cache)
- 2KB (only unified cache)
- 4KB
- 8KB
- 16KB
- 32KB
- 64KB

The L1 instruction cache, data caches, and unified cache store the valid bits for each cache line in RAM. The Cortex®-M52 processor provides a hardware mechanism to invalidate the cache at reset. This mechanism can be disabled to maintain valid cache state across reset, for example, where the RAM supports data retention and the processor logic is reset after powerup.

The automatic invalidation sequence can take a large number of cycles and executes independently of the instructions that are running on the processor. While the automatic invalidation sequence is in progress, any cache maintenance operation is treated as a NOP and instructions and data accesses do not look up in the cache. A DSB instruction waits for all automatic cache invalidate sequences to complete.

Software can also be used to perform a complete invalidation before enabling the data cache and unified cache on reset. The L1 instruction cache can be invalidated by a single instruction but the L1 data cache and unified cache need a loop iterating through all entries.

The architecture specifies the cache maintenance operations which can be used by software. The Cortex®-M52 processor includes memory-mapped registers that allow software to examine the content of the cache tag and data RAMs directly. This can be used for profiling or debugging the cache content. See [Direct cache access registers](#) for more information. The Direct Cache Access registers are only accessible in Secure state. Therefore, there is no requirement to restrict cache readability. The processor supports direct access to the cache RAM, therefore, access to the L1 instruction cache must also be restricted if XOM is supported in your system. This can be achieved by asserting the external input signal LOCKDCAIC. For more information on LOCKDCAIC, see [Miscellaneous signals](#).

Dirty data must be written back to external memory before the processor and RAM are powered down because the L1 data cache supports write-back operation.

All cache RAMs are standard single-ported RAMs and can be generated using standard RAM compilers.

### 10.10.1 L1 data cache

The Cortex®-M52 processor L1 data cache has the following features:

- It is a four-way set-associative cache.
- It has a cache line size of 32 bytes.
- It supports the following inner memory attributes and allocation hints for Non-shareable memory:
  - Write-Back and Write-Through Cacheable.
  - Read-Allocate and No Read-Allocate.
  - Write-Allocate and No Write-Allocate.
  - Transient and Non-transient. Clean cache lines that are associated with Transient memory are prioritized for eviction over lines that are associated with Non-transient memory.

Allocation into the L1 data cache depends on inner memory attributes only.

- The outer and inner memory attributes are exported on the AXI Main interface to support further system-level caching.
- The outer memory attributes are exported on the AHB Main interface to support further system-level caching.
- The Shareability attribute forces the region to be treated as Non-cacheable, regardless of the inner memory attributes. This enables maintaining coherency at the system-level.

Software or a debugger might use the direct cache access registers to read the contents of RAM arrays. The data cache is logically organized into two sets of RAM arrays. The dimensions of these RAM arrays vary with the cache size, the inclusion of Security Extension, and the inclusion of *Error Correcting Code* (ECC) logic.

**Table 10-9: Data cache RAM organization**

Array	Number of cache instances	Data stored	Write granularity	Array width excluding ECC (bits)		Array width including ECC (bits)		Array depth (number of entries)	
				4KB	64KB	4KB	64KB	4KB	64KB
Tag without security	4	Tag, valid, line status	RAM word	26	22	33	29	32	512
Tag with security		Tag, valid, line status, security		27	23	34	30		
Data	4	Data	Byte	32	32	39	39	256	4096

#### 10.10.1.1 No Write-Allocate mode

When a memory region is marked as Cacheable Write-Allocate, it normally allocates a cache line on a write miss. However, there are some situations where allocating on writes is undesirable, such

as executing the C standard library `memset()` function to clear a large block of memory to a known value.

Writing large blocks of data like this can pollute the cache with unnecessary data. It can also waste power and performance if a linefill must be performed only to discard the linefill data because the entire line was subsequently written by the `memset()`.

To prevent this, the Cortex®-M52 data cache includes logic to automatically disable data cache allocation on a write miss when streaming behavior is detected. When in this mode, writes are buffered and then written directly out to the external system through the Main interface even if they are cacheable.

No Write-Allocate mode is enabled when the data cache detects that three consecutive linefills have been overwritten by write data before being allocated to the cache. When enabled, the processor remains in No Write-Allocate mode until either:

- A linefill is allocated where a store has not overwritten a read from the Main interface.
- A linefill is started on an address which hazards on a buffered write or an outstanding write to the Main interface, indicating that it is unlikely to be related to the write data stream.

No Write-Allocate mode can be disabled by setting the `ACTLR.DISNWAMODE` to 1.

For more information on ACTLR, see [ACTLR, Auxiliary Control Register](#).

## 10.10.2 L1 instruction cache

The Cortex®-M52 processor L1 instruction cache has the following features.

- It is a two-way set-associative cache.
- It has a cache line size of 32 bytes.
- It does not allow writes to be performed, except for allocations.
- It only supports Read-Allocate for Inner Cacheable memory. Write-Allocate, Write-Back, Write-Through, and Transient attribute hints are ignored. Allocation into the L1 cache depends on inner memory attributes only.
- The outer and inner memory attributes are exported on the AXI Main interface to support further system-level caching.
- The outer memory attributes are exported on the AHB Main interface to support further system-level caching.
- The Shareability attribute is ignored for instruction side accesses.
- The Inner Cacheability attributes are always respected.

Debug accesses from the *Debug AHB* (D-AHB) slave interface on the processor cannot read information from the instruction cache.

Software or a debugger must use the direct cache access registers to read the contents of RAM arrays. The instruction cache is logically organized into two sets of RAM arrays. The dimensions of these RAM arrays vary with the cache size and the inclusion of *Error Correcting Code* (ECC) logic.

**Table 10-10: Instruction cache RAM organization**

Array	Number of cache instances	Data stored	Write granularity	Array width excluding ECC (bits)		Array width including ECC (bits)		Array depth (number of entries)	
				4KB	64KB	4KB	64KB	4KB	64KB
Tag	2	Tag and valid	RAM word	22	18	28	24	64	1024
Data	2	Instructions	RAM word	32	32	38	38	512	8192

### 10.10.3 L1 unified cache

The Cortex®-M52 processor L1 unified cache has the following features.

- It is a two-way set-associative cache.
- Its size is configured through 3 parameters: `ICACHESZ`, `DCACHESZ`, and `UCACHE`.
- It has a cache line size of 32 bytes.
- Store is not allowed to be performed, except for allocations of load and fetch.
- It only supports Read-Allocate for Inner Cacheable memory. Write-Allocate, Write-Back, Write-Through, and Transient attribute hints are ignored. Allocation into the L1 cache depends on inner memory attributes only.
- The outer and inner memory attributes are exported on the AXI Main interface to support further system-level caching.
- The outer memory attributes are exported on the AHB Main interface to support further system-level caching.
- The Shareability attribute is ignored for instruction side accesses.
- The Inner Cacheability attributes are always respected.
- It supports cache maintenance operation. It reuses relevant registers of data cache. See [Cache maintenance operations](#) for more information.
- It supports cache direct access. It reuses relevant registers of data cache, including `DCADCLR` and `DCADCRR`. See [DCADCLR bit assignments](#) and [DCADCRR bit assignments when reading the unified cache tag RAM](#) for more information.
- It supports cache access control through `CCR` and `MSCR` registers. It reused relevant bit of data cache, including `CCR.DC` and `MSCR.DCACTIVE`. See [Accessing the caches](#) for more information.
- It supports Error Correcting Code (ECC) logic when the Cortex®-M52 processor is configured with ECC. It reuses Error Bank register of data cache, including `DEBR0` and `DEBR1`. See [DEBR0 and DEBR1, Data Cache Error Bank Register 0-1](#) for more information.

Debug accesses from the *Debug AHB* (D-AHB) slave interface on the processor can read information from the unified cache.

Software or a debugger must use the direct cache access registers to read the contents of RAM arrays. The unified cache is logically organized into two sets of RAM arrays. The dimensions of these RAM arrays vary with the cache size and the inclusion of *Error Correcting Code* (ECC) logic.

**Table 10-11: Unified cache RAM organization**

Array	Number of cache instances	Data stored	Write granularity	Array width excluding ECC (bits)		Array width including ECC (bits)		Array depth (number of entries)	
				1KB	64KB	1KB	64KB	1KB	64KB
Tag	2	Tag and valid	RAM word	24	18	30	24	16	1024
Data	2	Instructions	RAM word	32	32	38	38	128	8192

### 10.10.4 Cache maintenance operations

All cache maintenance operations are performed through word stores to the *Private Peripheral Bus* (PPB) space using the relevant PPB architectural registers.

The following table lists the cache maintenance operations that are associated with the relevant cache type.

**Table 10-12: Cache maintenance operations**

Operation	L1 cache type	Register
Invalidate all	Instruction cache	ICIALLU
Invalidate by address	Instruction cache, data cache, and unified cache	ICIMVAU, DCIMVAC
Invalidate by set/way	Data cache and unified cache	DCISW
Clean by address	Data cache and unified cache	DCCMVAU, DCCMVAC
Clean by set/way	Data cache and unified cache	DCCSW
Clean and invalidate by address	Data cache and unified cache	DCCIMVAC
Clean and invalidate by set/way	Data cache and unified cache	DCCISW

Cache maintenance operations require software to use barriers carefully to guarantee intended operation:

- A **DMB** instruction is required to guarantee that a cache maintenance operation does not affect previous memory accesses.
- A **DSB** instruction is required to guarantee completion of all outstanding cache maintenance operations and to guarantee that outstanding cache maintenance operations do not affect any subsequent memory accesses.
- An **ISB** instruction is required to guarantee that the effects of all completed cache maintenance operations are visible to subsequent instruction fetches.

For more information on these barrier instructions, see the *Arm®v8-M Architecture Reference Manual*.

Cache maintenance is required when changing security attribution of an address by either reprogramming the *Security Attribution Unit* (SAU) or changing the external *Implementation Defined Attribution Unit* (IDAU) mappings.

Cache maintenance operations are supported in both Secure and Non-secure state. Software operating in Non-secure state cannot change secure data. Therefore, the behavior of some operations in Non-secure state is:

- Data Cache Line Invalidate by Set/Way (DCISW) is promoted to Data Cache Line Clean and Invalidate by Set/Way (DCCISW)
- Data Cache Line Invalidate by Address to *Point of Coherency* (PoC) (DCIMVAC) and Data Cache Line Invalidate to *Point of Unification* (PoU) are both promoted to Clean and Invalidate the data cache line which includes the selected address.

The Non-secure invalidate operations are only promoted if the processor is configured with the Secure extension. The promotion is only applied to the L1 data cache.

There are no data cache maintenance operations that operate on the entire cache or the unified cache. However, the processor provides a mechanism to automatically invalidate the cache at reset to initialize the structure before use.

Software can implement operations across the entire data cache by using the set/way operations to iterate across all the sets and ways of the cache. This method can also be applied to the unified cache.

For more information on cache maintenance operations, see the *Arm®v8-M Architecture Reference Manual*.

### 10.10.5 Automatic cache invalidation at reset

If the L1 caches move from an unpowered to a powered state, the caches are automatically invalidated. Automatic invalidation is also initiated when the RAM power domain is powered up when the core power domain is already active. For example, if the cache is re-enabled after it was shutdown to save power when not in use.

A small counter starts at the bottom of the caches and invalidates one line at a time. Until the automatic invalidation completes, any cache maintenance operation is treated as a NOP, no cache lookup or allocate is performed, and all data accesses to Normal Cacheable memory are effectively treated as Non-cacheable.

The automatic invalidation does not occur on transition to, or from, a cache retention state when controlled by the P-Channel interface. Automatic cache invalidation at reset can be disabled through the INITL1RSTDIS top-level input signal. Tying INITL1RSTDIS to 1, allows cache state to be maintained across reset. This can be used when the processor integration does not support power control using the P-Channel interface and the cache RAM supports state retention.



The invalidation sequence executes independently of the instructions running on the processor and is significantly more efficient than the equivalent software sequence. The instruction and data cache are invalidated in parallel with all cache ways invalidated simultaneously (two instruction cache lines and four data cache lines per cycle). In this process, the unified cache behaves in the same way as the instruction cache.



- While the automatic invalidation sequence is in progress, any cache maintenance operation is treated as a NOP and instruction and data accesses do not look up in the cache.
- If a DSB instruction is executed while the automatic invalidation sequence is in progress the instruction stalls the processor until the sequence is completed. The DSB can be interrupted if an exception of sufficient priority is pending and the automatic invalidation sequence continues. For more information on the instruction, see the *Arm®v8-M Architecture Reference Manual*.

The L1 data cache supports write-back operation. Therefore, dirty data must be written back to external memory before the processor and RAM are powered down. The processor provides register fields MSCR.DCACTIVE and MSCR.DCCLEAN to carry out this procedure.

For more information on MSCR, see [MSCR, Memory System Control Register](#).

## 10.10.6 Cache coherency

The Cortex®-M52 processor does not support hardware coherency for the L1 instruction cache, data cache, or unified cache. Coherency can only be maintained at the system level.

The following table summarizes the cache coherency usage models that the L1 data cache supports. The L1 instruction cache always follows the programmed Cacheability attributes and it is unaffected by the Shareable attribute that is defined in MPU\_RBAR.SH for the MPU region that is associated with an address. For more information on MPU\_RBAR, see the *Arm®v8-M Architecture Reference Manual*.

Further levels of caches are also supported.

For more information on further levels of caches, see [System cache support](#).

**Table 10-13: Coherency usage models available on the Cortex®-M52 processor**

MPU_RBAR.SH	Scenario description for L1 data cache
0b10, 0b11	<ul style="list-style-type: none"> <li>• All shareable locations are treated as inner Non-cacheable.</li> <li>• Programmed inner Cacheability attributes are ignored.</li> <li>• The L1 data cache is transparent to software for these locations. Therefore, no software maintenance is required to maintain coherency.</li> </ul>
0b00	<ul style="list-style-type: none"> <li>• Programmed inner Cacheability attributes are considered.</li> <li>• Data is not shared with other agents. Therefore, coherency issues do not exist.</li> </ul>



Note

For the programmed Cacheability attributes and the Shareability attribute defined in MPU\_RBAR.SH, the following effects exist:

- The attributes do not affect the L1 instruction cache.
- For the L1 unified cache, the attributes do not affect accesses from instruction fetch side, and only affect the load accesses from LSU.

## 10.10.7 Accessing the caches

If the Cortex®-M52 processor has been configured to include an instruction cache or data cache or unified cache, the CCR and MSCR registers are responsible for controlling access to the caches.

The following register bits are responsible for cache access:

- CCR.DC and CCR.IC are cache enable bits for the data cache/unified cache, and instruction cache respectively. If these bits are set to 0, then cache allocation is not allowed. Loads and stores can lookup and hit in the cache. Cache maintenance operations and direct cache accesses work normally.
- MSCR.DCACTIVE and MSCR.IACTIVE control cache access for the data cache/unified cache, and instruction cache respectively. If these bits are set to 0, then load and stores do not lookup or hit in the cache, and cache maintenance operations and direct cache accesses do not access the cache. These bits also serve as a hint to the system to indicate that power can be removed from the cache.

The following table describes the different cache access scenarios.

**Table 10-14: Cache access scenarios**

CCR	MSCR	Cache access behavior
CCR.DC and CCR.IC are set to 1	MSCR.DCACTIVE and MSCR.IACTIVE are set to 1	Normal operating mode. Unless PDCORE goes OFF resulting in PDRAMs going to RET, the caches are powered up and cache accesses can perform allocation and lookup.
CCR.DC and CCR.IC are set to 0	MSCR.DCACTIVE and MSCR.IACTIVE are set to 1	Cache lookups are allowed, but cache allocation is not permitted. This behavior is used to clean the cache before powering down.
CCR.DC and CCR.IC are set to 0 or 1	MSCR.DCACTIVE and MSCR.IACTIVE are set to 0	The caches are not being used, and they can be powered down. The CCR.DC and CCR.IC bits are ignored.



Note

- For more information on CCR, see the *Arm®v8-M Architecture Reference Manual*.
- For more information on MSCR, see [MSCR, Memory System Control Register](#).
- For more information on PDCORE and PDRAMs, see [Power domains](#).

## 10.10.8 System cache support

The following table shows the two optional levels of cache that the architecture implicitly defines.

**Table 10-15: System cache levels supported by Arm®v8.1-M and Cortex®-M52**

Cache level	Implemented by	Controlled by
L1	Internal processor caches	Inner Cacheability attributes
System level (L2)	External L2 cache controller integrated on the <i>Main Interface</i> .	<p>Outer Cacheability attributes</p> <p><b>Note:</b> For M-AXI, the Outer Cacheability attributes are exported, and the L2 cache controller uses the ARCACHE and AWCACHE signals to determine these attributes. For more information on these signals see <a href="#">M-AXI interface signals</a>. The ARINNER and AWINNER signals, which define the Inner Cacheability attributes can be used as hints for the L2 cache controller to optimize allocation or caching policy. The ARINNER and AWINNER signals can be used for debugging and monitoring purposes.</p> <p>For M-AHB, the Outer Cacheability attributes are exported, and the L2 cache controller uses the <i>HProt[6:0]</i> to determine these attributes.</p>

## 10.10.9 Direct cache access

The Cortex®-M52 processor provides a mechanism to read the embedded RAM that the L1 data, instruction, and unified caches use through **IMPLEMENTATION DEFINED** system registers. This functionality is useful to investigate data coherency issues.

There are four direct cache access registers:

- The read registers, DCADCRR and DCAICRR, for the L1 data and instruction cache respectively.
- The location registers, DCADCLR and DCAICLR, for the L1 data and instruction cache respectively.
- The unified cache reuses register of L1 data cache. When the unified cache is included, DCADCRR follows the register format of DCAICRR, and DCADCLR follows register format of DCAICLR.

Direct cache access registers are only accessible from the Secure privileged state, unless the processor core is configured without the Security Extension.



Note

- For more information on DCADCRR and DCAICRR, see [DCAICRR and DCADCRR, Direct Cache Access Read Registers](#).
- For more information on DCADCLR and DCAICLR, see [DCAICLR and DCADCLR, Direct Cache Access Location Registers](#).

## Reading a cache location

To read a cache location, the following steps must be performed in order:

1. The cache location to be read is written to the appropriate location register.
2. A read is then performed to the corresponding read register. This returns the data from that cache RAM location.

The location that is specified must be a physical RAM address. The processor translates the cache way into the appropriate RAM bank. The logical cache way and the physical RAM bank can be different because of the internal organization of the cache.

### Example code sequence for reading an instruction cache location

```
DCAICLR EQU 0xE001E214 ; Direct Cache Access Instruction cache Location
Register address
DCAICRR EQU 0xE001E204 ; Direct Cache Access Instruction cache Read Register
address

MOV R3, 0x0           ; Start building the value to write into the DCAICLR
                       ; Bit[0]==0b0, to target the tag RAM
LSL R0, #5
ORR R3, R0             ; Put the cache index into bits[14:5] of DCAICLR

LSL R1, #31
ORR R3, R1             ; Put the way into bit[31] of DCAICLR

LDR R11, =DCAICLR
STR R3, [R11]          ; Write the location into DCAICLR

LDR R11, =DCAICRR
LDR R4, [R11]          ; Read DCAICRR, R4 will be updated with the contents of the
Instruction cache tag
                       ; at the supplied index and way
```

## ECC errors

Direct accesses ignore all *Error Correcting Code* (ECC) errors and cannot be used to read the ECCs in the RAMs.

## Accessing a cache location

For details on the encoding of the DCADCRR and DCAICRR registers, see [DCAICRR and DCADCRR, Direct Cache Access Read Registers](#).

When the data RAM is specified in either the DCADCRR[0] or DCAICLR[0], the data offset field determines the word that is read which is in DCACLR[5:1].

When the tag RAM is specified in DCADCRR[0], DCAICLR[0], or DACDCLR[0], the tag encoding that is written to DCADCRR, DCAICRR or or DCADCRR for the data, instruction, and unified caches respectively is shown in the following tables. Unused fields in the data register are written as zero.

**Table 10-16: DCADCRR data format for data cache tag RAM reads**

Cache size	Nonsecure bit	Status bits	Valid bit	Tag bits
4KB	[26]	[25:23]	[22]	[21:0]

Cache size	Nonsecure bit	Status bits	Valid bit	Tag bits
8KB	[26]	[25:23]	[22]	[21:1]
16KB	[26]	[25:23]	[22]	[21:2]
32KB	[26]	[25:23]	[22]	[21:3]
64KB	[26]	[25:23]	[22]	[21:4]

**Table 10-17: DCAICRR data format for instruction cache tag RAM reads**

Cache size	Valid bit	Tag bits
4KB	[23]	[22:2]
8KB	[23]	[22:3]
16KB	[23]	[22:4]
32KB	[23]	[22:5]
64KB	[23]	[22:6]

**Table 10-18: DCADCRR data format for unified cache tag RAM reads**

Cache size	Valid bit	Tag bits
1KB	[23]	[22:0]
2KB	[23]	[22:1]
4KB	[23]	[22:2]
8KB	[23]	[22:3]
16KB	[23]	[22:4]
32KB	[23]	[22:5]
64KB	[23]	[22:6]

The Non-secure bits in the table only applied to data cache with Security Extension.

The STATUS bits in the data cache tag RAM contain information regarding:

- The clean/dirty status.
- Arm®v8.1-M transient attribute for a valid cache line.
- Outer attributes for a valid cache line.

For more information on the STATUS bits, see [DCAICRR and DCADCRR, Direct Cache Access Read Registers](#).

The following table describes the information that is stored in a state-dependent format.

**Table 10-19: Data cache tag RAM status encoding**

Status encoding	Line Clean/Dirty	Line Transient	Outer attributes
0b000	Clean	Yes	UNKNOWN
0b001	Clean	No	UNKNOWN
0b010	Dirty	No	Non-cacheable
0b011	Dirty	No	Write-Back, Write-Allocate

Status encoding	Line Clean/Dirty	Line Transient	Outer attributes
0b100	Dirty	No	Write-Back, No Write-Allocate
0b101	Dirty	No	Write-Through, Write-Allocate
0b110	Dirty	No	Write-Through, No Write-Allocate



Note

- 0b111 is reserved.
- Outer attributes are only valid for lines allocated to Inner write-back memory regions when they are made dirty by a write.
- Only clean lines can be distinguished as transient. When a line has been written as dirty, it is evicted from the cache by a subsequent line-fill with the same priority as other non-transient lines.

## 10.11 Store buffer

The memory system includes a *Store Buffer* (STB) to hold data before it is written to the cache RAMs or passed to the Main interface. All store instructions to Normal memory regions that are not the *Tightly Coupled Memory* (TCM), *Private Peripheral Bus* (PPB), or *Peripheral-AHB* (P-AHB) interface must pass through the STB.

When data cache is included, the STB has four identical slots which hold the address, up to 32 bits of data, and other attributes of store transactions. Otherwise, the STB has only two identical slots.

### 10.11.1 Store buffer merging

The *Store Buffer* (STB) has merging capabilities. If a previous write access has updated an entry, other write accesses on the same word can merge into this entry. Merging is only possible for stores to Normal memory.

Merging is not possible if:

- The access is to Device memory.
- The first access leaves the STB, either on the Main Interface or to the cache, before the second access reaches the STB.
- There is an attribute or security mismatch.
- Either access is a Store-Exclusive.
- The second access is a Store-Release.

### 10.11.2 Store buffer behavior

The *Store Buffer* (STB) directs cacheable write requests to the cache controller and Main interface blocks.

#### Cache controller for cacheable write hits

The store buffer sends a cache lookup to check that the cache hits in the specified line, and if so, the store buffer merges its data into the cache when the entry is drained.

#### Main interface

For Non-cacheable, and Cacheable No Write-Allocate stores that miss in the L1 data cache, a write access is performed on the Main interface.

For Cacheable Write-Allocate stores that miss in the data cache, a linefill is started using linefill buffers. The store data is sent to the linefill buffer first, and then the Main interface data is merged.

### 10.11.3 Store buffer ordering

The *Store Buffer* (STB) has ordering capabilities and must maintain ordering between some stores.

The STB ordering is compulsory for the following stores:

- All Device stores must occur in order with respect to other Device accesses.
- Stores after a load-acquire must occur after the load-acquire.
- Stores before a store-release must occur before the store-release.

### 10.11.4 Store buffer draining

The *Store Buffer* (STB) is drained of all stores to Device memory before a load is performed from Device memory.

Slots that are Non-mergeable drain quickly because there is no benefit in being present in the STB. Mergeable slots might wait for future stores to merge into them and reduce the number of cache writes required.

A store buffer entry is drained if:

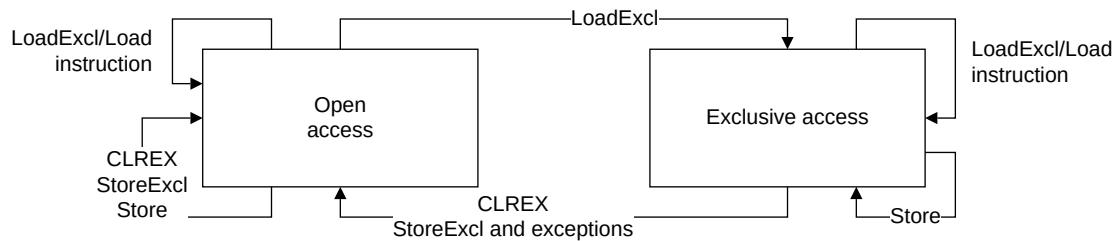
- There is a cache maintenance operation pending.
- There is a store that cannot enter the STB because of the current contents of the STB.
- There is a DSB, DMB, or ESB instruction.
- There are debug events.

## 10.12 Internal local exclusive access monitor

The Cortex®-M52 processor implements an internal local exclusive access monitor that does not tag addresses. This implies that the reservation granule is the entire memory.

The following figure shows the operation of the internal local exclusive monitor, including all **IMPLEMENTATION DEFINED** options.

**Figure 10-2: Operation of internal exclusive access monitor**



- LoadExcl are exclusive load instructions to addresses associated with the *Tightly Coupled Memory* (TCM), Main interface, and *Peripheral AHB* (P-AHB) interfaces which are either Non-shareable or Shareable when the system supports a global exclusive monitor.
- Exclusive Load instructions which access addresses in the *Private Peripheral Bus* (PPB) region, including the *Internal Private Peripheral Bus* (IPPB) registers and the *External Private Peripheral Bus* (EPPB) interface do not update the internal exclusive monitor.
- Exclusive Load instructions do not update the internal exclusive monitor if these instructions are in Shareable memory addresses associated with the Main interface and P-AHB interfaces where a global exclusive monitor is not supported.
- Exclusive Store instructions (StoreExcl) always clear the internal exclusive monitor.
- *AHB TCM Access* (TCM-AHB) accesses to TCM do not affect the internal local exclusive access monitor. There is no hardware support for concurrency control between software and TCM-AHB to TCM.
- *Memory Built-In Self Test* (MBIST) and *Debug AHB* (D-AHB) accesses do not affect the internal local exclusive access monitor.
- Exception entry and return are architecturally defined to clear the local exclusive access monitor.



## 10.13 Main interface and P-AHB interaction with the global exclusive monitor

The Main interface and *Peripheral AHB* (P-AHB) interfaces support systems that include a global exclusive monitor by using the interface signals that conform to the AMBA® 5 AXI and AMBA 5 AHB protocols respectively.

Accesses associated with load and store exclusive instructions are only handled as exclusive on the M-AXI/M-AHB and P-AHB interfaces if they are either of the following:

- Device memory.
- Normal memory marked as Shareable in the associated *Memory Protection Unit* (MPU) region.

Exclusive accesses to Normal Shareable memory are always treated as Shareable Non-cacheable by the processor.

Only the internal exclusive access monitor handles accesses to Non-shareable memory.

If an Exclusive read access is carried out to a region that does not support a global exclusive monitor, the slave must respond in either of the following ways:

- An OKAY response for M-AXI.
- The HEXOKAY response must be deasserted for M-AHB.
- The HEXOKAYP response must be deasserted for P-AHB.

These responses do not result in the processor taking an exception, but they do ensure that the STREX does not pass. This kind of livelock behavior can be trapped using a Watchdog unit.



Note

The default memory map includes only Non-shareable Normal memory regions. Therefore, Cortex®-M52 processor configurations without an MPU can only generate external exclusive load and store operations from Device memory in either the Peripheral region, External Device region or Vendor\_SYS region. For more information on the memory map, see [Memory map](#)

## 10.14 MBIST

The Cortex®-M52 processor supports two *Memory Built-In Self-Test* (MBIST) use models.

### Production MBIST

This allows memory testing during manufacture. This use model requires that a production MBIST controller is inserted into the processor and connected to the internal MBIST interface. This can be automatically carried out by EDA tools using configuration information that is delivered with the processor.

## On-line MBIST

On-line MBIST allows memory and *Error Correcting Code* (ECC) logic testing during functional operation. The optional *Programmable MBIST Controller* (PMC-100) supports on-line MBIST and is integrated into the processor. This use model can be used to support fault detection and analysis as a part of a functional safety environment.

The Cortex®-M52 processor supports direct access to the embedded RAM associated with the L1 Instruction and Data cache and Unified cache and the TCM while the processor is operational. This feature, called On-line MBIST operation, can be used to test the ECC logic and maintain the RAM during runtime with minimum impact on the performance of software. Typically, uses of on-line MBIST include:

- Analysis of Errors including categorization into transient (soft) or permanent (hard) errors
- Memory Scrubbing – correcting errors in the RAM to prevent accumulation and to reduce the probability of escalation to uncorrectable errors
- Error injection for testing of error management software. On-line MBIST is managed by a dedicated component embedded in the Cortex®-M52 processor, the PMC-100.

The PMC-100 is configured at implementation by setting the Verilog parameter `PMC`. The component is programmed through memory mapped registers in the *Private Peripheral Bus* (PPB) region of the memory map based at address `0xE0046000`. The PMC-100 contains CoreSight ID registers and will be listed in the processor ROM table when configured. If the Cortex®-M52 processor is configured with the Arm®v8.1-M Security extension the PMC-100 can only be programmed by software running in Secure privileged state, or by the debugger when Secure debug is enabled in the system. All accesses to PMC-100 registers from Non-secure state will be treated as RAZ/WI and all unprivileged accesses will raise a BusFault exception. The processor also supports direct access to the PMC-100 from an external agent in the system through an AMBA APB4 slave interface. Access to the PMC-100 on this interface is only permitted for requests marked as secure and privileged in PMCPPROT. The PMC-100 is an optional processor component delivered as part of the Cortex®-M52 Safety Package.



The Cortex®-M52 processor does not support an external MBIST interface.

---

# 11. Reliability, Availability, and Serviceability Extension support

This chapter describes the *Reliability, Availability, and Serviceability* (RAS) features implemented in the Cortex®-M52 processor.

## 11.1 Cortex®-M52 processor implementation of RAS

The Cortex®-M52 processor implements the Arm®v8.1-M *Reliability, Availability, and Serviceability* (RAS) features to ensure correct operation in environments where functional safety and high-availability are critical. The RAS Extension is always included in the Cortex®-M52 processor, however most of the features are only supported when *Error Correcting Code* (ECC) is configured and enabled.

The Cortex®-M52 processor standardizes the software interface for fault detection and analysis by supporting the RAS Extension. The RAS features supported are *Error Correcting Code* (ECC) for the L1 instruction cache and data cache, unified cache, and TCMs.

Errors are reported to the system through:

- Output signals on the processor.
- Error bank registers which can be used to mitigate hard errors that cannot be corrected by writing back to the RAM. For more information, see [Error bank registers](#).
- The architectural registers that are defined by the RAS Extension. For more information, see [RAS Extension registers](#)

### Supported RAS architectural features

The RAS architecture contains:

- An *Error Synchronization Barrier* (ESB) instruction.
- An implicit ESB operation that is inserted after exception entry, exception return, and lazy stacking. This feature is enabled by setting AIRCR.IESB. For more information on AIRCR, see the *Arm®v8-M Architecture Reference Manual*.
- Two ID registers, ERRDEVID and ID\_PFR0. For more information on these registers, see the *Arm®v8-M Architecture Reference Manual*.
- A fault status register, RFSR, that is dedicated to RAS events. For more information on:
  - RAS events, see [Cortex-M52 RAS events](#).
  - RFSR, see [RFSR, RAS Fault Status Register](#).
- A summary register indicating the nodes that have detected RAS events, ERRGSR. For more information on this register, see [ERRGSR0, RAS Fault Group Status Register](#). A node is a unit that can detect RAS events, and for Cortex®-M52, a node is the entire processor. Therefore, all RAS events are logged in the same location and the processor supports a single error record.

- Each node has one set of Error Record Registers that can store information about the last RAS event that the node has detected.  
The RAS Error Record Registers are independent of the Error Bank Registers, although they have some common behavior. Either or both of the register types can be used by system software that is handling errors. However, for compatibility across other devices and systems that implement the RAS Extension, the RAS programmers' model must be considered. The RAS Error Record Registers are described in [RAS Extension registers](#) and the Error Bank Registers are described in [Error bank registers](#).



For a complete description of RAS error types and the information on RAS errors that are produced at the node, see the *Arm® Reliability, Availability, and Serviceability (RAS) Specification*.

### 11.1.1 Cortex®-M52 RAS events

The *Reliability, Availability, and Serviceability (RAS) Extension* provides a standard model for recording and reporting errors which might occur during the operation of a system.

In the Cortex®-M52 processor, the following are considered as RAS events:

- L1 instruction cache or unified cache *Error Correcting Code (ECC)* errors.
- L1 data cache ECC errors.
- TCM ECC errors.



For more information on how these RAS events are detected and handled in the Cortex®-M52 processor, see [ECC memory protection behavior](#) to get an overview on how instruction cache, unified cache, data cache, and TCM ECC errors are handled.

## 11.2 ECC memory protection behavior

*Error Correcting Code (ECC)* memory protection is optional. At implementation, you can configure the Cortex®-M52 processor to include ECC or not using the Verilog parameter, `ecc`. At Cold reset, if the Cortex®-M52 processor is configured with ECC, you can control whether ECC is enabled or not using the static configuration signal `INITECCEN`. `INITECCEN` must only be changed when the processor is powered down and in Cold reset.

ECC memory protection includes the following protection features:

- Data protection
- Address decoder protection

- White noise protection, which involves protection against faults in the RAM that might also result in no entry being selected and therefore, resulting in reading either all zeros or all ones.

### 11.2.1 ECC schemes and error type terminology

The Cortex®-M52 processor supports two *Error Correcting Code* (ECC) schemes to detect errors.

#### ECC schemes

##### SECEDED

*Single Error Correct Double Error Detect* (SECEDED) is used on the L1 data cache and TCM RAMs. The SECEDED scheme also provides information on how to correct the error.

##### DED

*Double Error Detect* (DED) is used on the L1 instruction cache or unified cache RAMs. The DED scheme detects single bit and double bit errors. The instruction cache or unified cache does not need a correction mechanism or scheme because the contents must always be consistent with external memory. Therefore, the processor automatically invalidates the instruction cache or unified cache RAM to correct its contents.

In the Cortex®-M52 processor, the ECC schemes can also support detection of some multi-bit errors where more than two bits are incorrect. Where possible, RAM location information is included in the ECC code to allow fault detection in the RAM address decoder logic.

#### Error type terminology

The following error type terminology is used in this manual in the context of ECC:

##### Single-bit error

An error where only one bit of the data or ECC code is incorrect. These errors can usually be corrected.



ECC errors detected in the address field are treated as multi-bit errors, because this indicates that an incorrect location has been read and all of the data is wrong.

---

##### Multi-bit error

An error in which any one of the following is true:

- More than one bit of data or ECC code is incorrect.
- An error is detected in one or more address bits.
- The RAM read value is all ones or all zeros.

##### Corrected error (CE)

An ECC error that is detected by hardware and that hardware can correct. These are:

- Single bit errors, which can be corrected inline by flipping the faulty bit.

- All errors which can be corrected by refetching the data from external memory. This includes all instruction cache or unified cache errors and all data cache errors when the cache line can be guaranteed to be clean.

For more information on Corrected errors (CEs), See *Arm® Reliability, Availability, and Serviceability (RAS) Specification*.

### Uncorrected error (UE)

An ECC error that cannot be corrected or deferred. These are multi-bit errors:

- From the TCMs.
- In an L1 dirty data cache data RAM where it is not guaranteed that the cache line is clean. This includes the case where the ECC indicates that the RAM location is incorrect.
- In an L1 dirty data cache tag RAM where it is not guaranteed that the cache is clean. This includes the case where the ECC indicates that the RAM location is incorrect.

For more information on Uncorrected errors (UEs), see *Arm® Reliability, Availability, and Serviceability (RAS) Specification*.

## 11.2.2 Enabling ECC

If configured in the processor, *Error Correcting Code* (ECC) is enabled at Cold reset using the input signal INITECCEN.

For more signal information, see [Error interface signals](#). For more information on MSCR, see [MSCR, Memory System Control Register](#).

If ECC is enabled out of reset, the L1 cache must be invalidated before it is enabled to avoid spurious ECC errors being detected because of a mismatch between the data and ECC in the RAM. Automatic instruction and data cache and unified cache invalidation can be enabled at reset by tying the input signal INITL1RSTDIS LOW. For more signal information, see [Error interface signals](#). For more information on automatic cache invalidation, see [Automatic cache invalidation at reset](#).

Spurious ECC errors from speculative read and sub-word write requests to uninitialized TCM at start-up can be avoided using MSCR.TECCCHKDIS. Setting this field disables ECC checking, correction, and reporting so the memory and error correction code can be safely initialized by software.



Software can determine whether ECC is configured and enabled by reading MSCR.ECCEN. However, software cannot enable ECC.

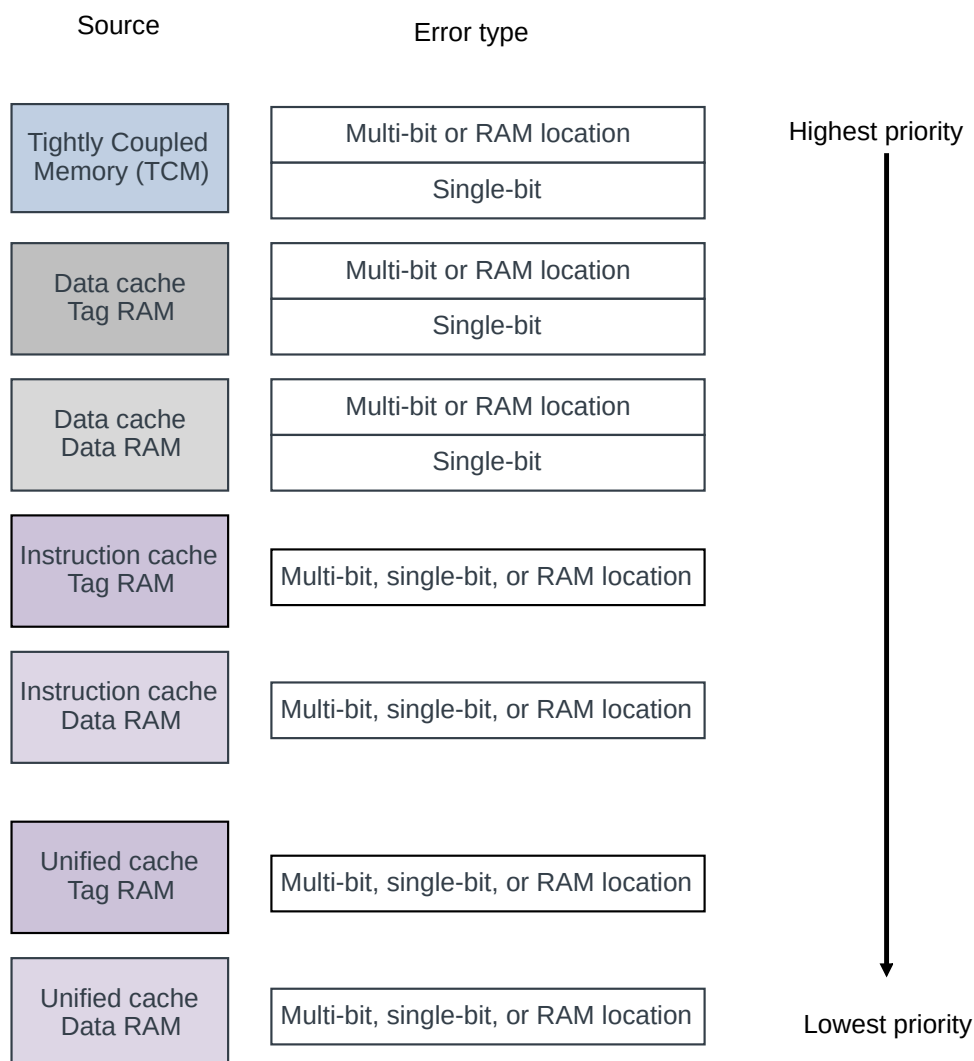
---

## 11.2.3 Error detection and processing

The Cortex®-M52 processor core is responsible for error detection and processing. Multiple errors can occur simultaneously, therefore, the processor prioritizes the error processing based on the source.

The following figure shows the prioritization of error processing that occurs in the order of decreasing priority.

**Figure 11-1: Error processing prioritization**



The errors in the *Data Tightly Coupled Memory* (DTCM) always have higher priority than the errors in the *Instruction Tightly Coupled Memory* (ITCM).

### 11.2.3.1 Error processing in the L1 data, instruction, and unified cache

The cache tag and data RAMs are read during various operations that the Cortex®-M52 processor carries out.

The following table lists these operations.

**Table 11-1: L1 cache RAM access classes**

Access type	RAM block read	Notes
Instruction fetch or unified cache instruction fetch and load	Instruction tag and data RAM	Two tag banks and up to two data banks
Load request	Data tag and data RAM	4 tag banks and up to four data banks
Dirty line eviction	Data RAM	Entire line is read in two cycles, half line per cycle.
Store buffer address read	Data tag RAM	Four tag banks
Store buffer data read	Data data RAM	Only used for <i>Read-Modify-Write</i> (RMW). RMW is used when the processor writes a partial word when ECC is enabled. Store operations to a cache line, which are less than 32 bits of data must read the data RAM to construct the ECC to write back. This is based on the combination of the current and new data. This read operation can result in an error being detected in the data RAM.
Data cache maintenance	Tag RAM and data RAM	Tag RAM read for address-based and clean operations. Data RAM read for clean evictions.

The error processing operations are:

#### Instruction fetch in instruction cache or instruction fetch and load in unified cache

All *Error Correcting Code* (ECC) errors on instruction fetches in instruction cache or instruction fetches and loads in unified cache are processed by invalidating the tag RAM and refetching the line from external memory.

#### Corrected errors in the L1 data cache for load and store operations

*Corrected errors* (CE) in the L1 data cache that are detected on load, store, and cache maintenance operations are processed by cleaning (if required) and invalidating the location. For load operations, the data is corrected by replaying, which is refetching and executing the instruction, causing a data cache miss on the invalidated location and reading the correct data from external memory.

Store operations to Write-Allocate memory request a linefill after the error has been processed and then merge the write data into the line as it is allocated to the cache. Store



operations to a line in the cache which write less than 32 bits of data must read the data RAM to construct the ECC to write-back, based on a combination of the current and new data. This read operation can result in an error being detected in the data RAM.

### Cache maintenance operations

Data cache maintenance operations which operate on an address read all four tag RAMs to check for a match. Instruction cache maintenance operations which operate on an address read two tag RAMs to check for a match. Therefore, they can potentially detect multiple errors unrelated to the requested location. The operation automatically cleans and invalidates all detected errors in sequence. Cache maintenance invalidate by set/way location carried out by Non-secure code always reads the tag because it might contain a dirty line associated with a Secure address, and therefore, it must be cleaned to prevent data loss before being invalidated. The behavior of cache maintenance operations in Non-secure state is described in [Cache maintenance operations](#).

### Dirty line eviction

In all cases where a line is evicted, the data RAM associated with the entire line is read out of the cache. Any error detected in this read is corrected inline before being written back to the external memory through the Main interface. If a multi-bit error is detected in the data, the line is marked as poisoned and an imprecise BusFault is raised if MSCR.EVECCFAULT is set.

Multiple errors are processed according to the priority listed in [Error detection and processing](#). Errors during load operations are handled by replaying the instruction. Therefore, it is possible for errors found in multiple cache ways to not be processed if the original lookup is not repeated. For example, if the replayed load is interrupted.

If data is lost because of a multi-bit ECC error, then an Imprecise BusFault is generated under the following conditions:

- If a data cache eviction is performed, and a multi-bit error is detected in the data RAM and MSCR.EVECCFAULT is set.
- If a data cache line is invalidated because of a multi-bit error detected in the Tag RAM, and MSCR.DCCLEAN is not set.

Although loads do not directly cause BusFaults, they cause ECC maintenance behavior that triggers a BusFault if data is lost. Additionally, if any load sees an ECC error the pipe is flushed, and the load cannot progress until the ECC maintenance has finished. This guarantees that the core does not consume erroneous data until an Imprecise BusFault has been generated.

A multi-bit error on the data cache tag when MSCR.DCCLEAN is asserted is always correctable as the corresponding cache line cannot contain any dirty data.

A multi-bit error on the data cache data when MSCR.EVECCFAULT is deasserted is considered Deferred (DE), because when that line is evicted, it is marked as poisoned. MSCR.EVECCFAULT being deasserted implies that the system supports poisoning.

Any other case of multi-bit errors in the data cache is considered Uncorrected.

### 11.2.3.2 Error processing in the TCMs

Error detection and correction are carried out on each of the individual TCMs, that is, ITCM, DOTCM, and D1TCM. Accesses to each of the interfaces are treated in the following way:

- Correctable errors detected during instruction fetch and load operations result in the read being repeated either by refetching the instruction address or replaying the load instruction. The corrected data is written back to the TCM.
- Correctable errors from read requests on the *AHB TCM Access* (TCM-AHB) are corrected inline and returned to the system on completion of the transaction.
- Write requests to the TCM with an access size smaller than a complete word or with non-contiguous bytes from TCM-AHB or *M-profile Vector Extension* (MVE) operations must carry out a *Read-Modify-Write* (RMW) sequence to the TCM. Correctable errors detected during the sequence are corrected inline before the complete store word is written back to the TCM. Uncorrectable errors that are detected on the read phase of an RMW sequence cause the write phase to be abandoned, and the address is marked as poisoned in the error bank register. If the location is read again, a precise BusFault is raised.
- When ECC is enabled, an instruction fetch or load operations might raise a precise BusFault exception, if an *Uncorrected error* (UE) is detected.



When ECC is enabled, before performing a less than 32 bits of data write to a TCM location which causes an RMW, you must initialize the location first by performing an aligned word write to the location. Arm® recommends that all TCM locations are initialized in this manner by boot code.

### 11.2.4 Error reporting

Error reporting is done using both registers and output signals.

#### Corrected errors

Corrected errors (CE) are always transparent to program flow. For more information on Corrected errors (CEs), see *Arm® Reliability, Availability, and Serviceability (RAS) Specification*.

#### Uncorrected errors

Uncorrected errors (UEs) can result in a precise or imprecise BusFault. If an exception occurs, the source of the error can be determined using the AFSR and RFSR.

An imprecise BusFault is raised when a UE is found in the data cache data RAM during an eviction. If the system supports poisoning, clearing MSCR.EVECCFAULT disables this error. An imprecise BusFault is also raised when a UE is found in the data cache tag RAM and MSCR.DCCLEAN is not set and this type of BusFault cannot be disabled. For more information on Uncorrected errors (UEs), see *Arm® Reliability, Availability, and Serviceability (RAS) Specification*.

Errors detected on accesses to the TCMs never result in an imprecise BusFault.

## Errors on the L1 instruction cache, L1 data cache, unified cache, and TCMs

Errors detected in the L1 instruction cache, L1 data cache, unified cache, and TCMs are reported on the following external error interface output signals:

- DMEV0
- DMEV1
- DMEV2
- DMEL0[2:0]
- DMEL1[2:0]
- DMEI0[25:0]
- DMEI1[25:0]

Up to two errors can be reported on the same cycle. If multiple simultaneous errors occur, the priority scheme for reporting is followed. The reporting priority is described in [Error detection and processing](#). If up to two errors occur, the location and error class is indicated in DMELn and DMEIn respectively, and DMEVn is asserted. If more than two errors occur, then only information about the two highest priority errors are reported and DMEV2 is asserted to indicate further information is not available.

For more signal information, see [Error interface signals](#).



A particular ECC error might be reported multiple times on the DME bus.

---

## Error bank registers

The processor includes internal error bank registers which do the following:

- Record the two most recent errors detected.
- Isolate the system from hard errors in the RAM which cannot be corrected by invalidating or overwriting with correct data.

Two error bank registers are included for each source of errors:

- IEBR0 and IEBR1 for the L1 instruction cache.
- DEBR0 and DEBR1 for the L1 data cache or unified cache.
- TEBR0, TEBR1, TEBRDATA0, and TEBRDATA1 that are shared across the ITCM and DTCM.

## Error bank behavior

When an error bank contains a valid entry, any errors detected from the associated RAM address are ignored.

## L1 instruction and data cache and unified cache

For the L1 instruction and data cache and unified cache, the RAM addresses are masked on a cache lookup and no longer used for allocating a line on a miss, isolating the processor from any potential hard errors in the RAM which could cause incorrect behavior even if corrected data is written from external memory.

## TCMs

For TCMs, each TCM error bank contains a 32-bit data register `TEBRDATAn`. When a single-bit TCM fault is detected and the error bank is allocated, the corrected data is written to the data register and the TCM memory. Any subsequent read returns the result directly from `TEBRDATAn`. Writes to an address associated with a valid TCM Error bank is written to both the `TEBRDATAn` and the TCM RAM to maintain consistency if the error bank is reallocated or cleared by software. If a multi-bit error is detected on a read from the TCM RAM, the error bank `TEBRn.POISON` field is set. When this field has been set any subsequent read requests to the TCM which matches the error bank address, it will result in an error. A precise `BusFault` will be raised for a load request from the processor and `HRESP` is asserted on a read on the *AHB TCM Access* (TCM-AHB) interface.

Write accesses from store instructions or TCM-AHB to TCM that match an error bank register with `TEBRn.POISON` set do not raise a fault. The `TEBRn.POISON` field is cleared by an aligned 32-bit write to the address associated with the TCM error bank register. The behavior of the poison feature in the TCM error bank register allows hard multi-bit errors to be patched by software. For example:

1. Load from the TCM at an address detects a multi-bit *Error Correcting Code* (ECC) error. `TEBRn` is allocated, `TEBRn.POISON` is set, and a fault is raised.
2. Patch write data of 32 bits is stored to the TCM at that address. `TEBRDATAn` and TCM memory are updated and `TEBRn.POISON` is cleared.
3. Subsequent read and write transactions to that address are completed as expected.

If this sequence is applied, the failing TCM RAM entry is isolated and normal execution can continue when the write is applied, even when the error is Hard and so cannot be cleared by a patch directly to the RAM. Between steps 1 and 2, read and write transactions with size less than a word continue to raise a fault because the address has not been patched.

The error bank registers are updated when an ECC error from the associated RAM controller has been processed and remains valid until either a subsequent error is detected and processed, or a direct software write to the bank is carried out to clear the data.

Invalid error banks are always allocated in preference to valid error banks. If both error banks contain valid data new errors are allocated using a round-robin approach. Error banks can be locked from being overwritten by writing to the `LOCKED` field in the error bank register.

The error bank registers are only cleared on Cold reset and retain their content on system reset.

## 11.2.5 Address decoder protection and white noise protection

The Cortex®-M52 processor includes address decoder protection and white noise protection.

### Address decoder protection

Address decoder protection detects some of the errors that might occur because of a failure in the address decoder in a RAM instance. A fault in a RAM address decoder circuit might result in the wrong RAM entry being selected, which typically contains data and ECC that are self-consistent. Therefore, an ECC error on the data is not generated in this case, but the wrong data is read from the RAM.

### White noise protection

A fault in a RAM might result in no entry being selected, which might result in reading either all zeros or all ones. Protection against such faults is white noise protection.

## 11.3 Flop parity

The Cortex®-M52 processor can be configured to include extra logic to check the integrity of flip-flops in the functional (non-debug) logic in the presence of potential Single Event Upset faults (SEU).

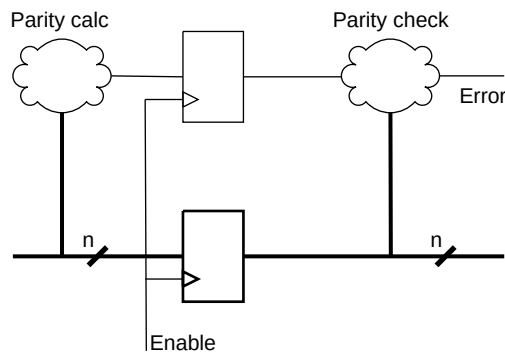
This option can provide additional fault coverage in safety-related applications. The aim of the design is to attain > 90% coverage of SEU faults to meet the requirements for the ISO 26262 ASIL-B *Single Point Fault Metric* (SPFM).

When included, this option instantiates additional logic to calculate parity for a group of flops that have a common enable term. The parity information is stored in an additional flop. The output of this flop is used to confirm the output of the original group as shown in the following figure. A difference in parity indicates an SEU has occurred on the design flops or on the parity flip-flop itself. The error signals from each set of parity logic are combined into the external output signal DFE. Flop parity is configured at implementation using the Verilog parameter `FLOPPARITY`.

**Table 11-2: Detected parity error from the flip-flop protection logic**

DFE[1:0]	Description
[1]	IWIC flip-flop parity error
[0]	PDCORE flip-flop parity error

**Figure 11-2: Parity logic associated with a group of design flops**



Flop parity operation requires that all flip-flops in the design are initialized to known values. This is achieved by setting the Verilog parameter RAR to 1 when FLOPPARITY is set to 1.



DCLS and flop parity are mutually exclusive processor options. If the Verilog parameter DCLS is set to 1, then the FLOPPARITY parameter must be set to 0. Likewise, if the FLOPPARITY parameter is set to 1, then the DCLS parameter must be set to 0.

## 11.4 Interface protection behavior

The Cortex®-M52 processor includes parity-based interface protection on the AXI Main (M-AXI)/AHB Main (M-AHB), Peripheral AHB (P-AHB), External Private Peripheral Bus (EPPB) interfaces and AHB TCM Access (TCM-AHB), Debug AHB (D-AHB), and PMC-100 APB (PMC-APB) slave interfaces.

This feature is configured at implementation time by setting the configuration parameter `BUSPROT`. Each interface includes side-channels on the control and data signals providing point-to-point protection between the processor and the interconnect. Odd parity is used to protect signals, with all data and address signals supported on an 8-bit granularity. The interface protection is designed to be used together with other processor and system level features to provide support for safety-related applications.

Interface protection on AXI is a super-set of the data check feature. `RDATACHK` and `WDATACHK` are considered part of the interface protection signal group. If interface protection is not configured in the processor, `RDATACHK` is unused and `WDATACHK` is tied to 0.

- [M-AXI interface protection signals](#)

- CODE-AHB interface protection signals
- SYS-AHB interface protection signals
- P-AHB interface protection signals
- EPPB interface protection signals
- TCM-AHB interface protection signals
- D-AHB interface signals
- PMC-100 interface signals

Parity is only checked for each signal on the interface when the signal is valid.

**Table 11-3: Parity checking conditions**

Interface	Parity checking conditions
M-AXI	<p>ACLKEN and AWAKEUP are always checked.</p> <p>For each channel (AR, AW, R, W, and B):</p> <ul style="list-style-type: none"> <li>• VALID and READY are checked when ACLKEN is HIGH.</li> <li>• The remaining signals in each channel (which carry the payload) are checked when the VALID signal for the channel and ACLKEN are both HIGH. When the VALID signal for the channel is HIGH, this indicates that the payload is valid according to the AXI protocol.</li> </ul>
M-AHBC	<p>HREADYC, HADDRRC, and HTRANSC are always checked.</p> <p>HBURSTC, HWRITEC, HSIZEC, HNONSECC, HEXCLC, HMASTERC, and HPROTC are checked when HTRANSC != IDLE.</p> <p>HWDATAC is checked in data phase for write transfer.</p> <p>HRDATAC is checked in data phase for read transfer when HREADYC == 1.</p> <p>HRESPC and HEXOKAYC are checked in data phase.</p>
M-AHBS	<p>HREADYSYS, HADDRSYS, and HTRANSSYS are always checked.</p> <p>HBURSTSYS, HWRITESYS, HSIZESYS, HNONSECSYS, HEXCLSYS, HMASTERSYS, and HPROTSYS are checked when HTRANSSYS != IDLE.</p> <p>HWDATASYS is checked in data phase for write transfer.</p> <p>HRDATASYS is checked in data phase for read transfer when HREADYSYS == 1.</p> <p>HRESPSYS and HEXOKAYSYS are checked in data phase.</p>
P-AHB	<p>HTRANSP, HADDRP, and HREADYP are always checked.</p> <p>HBURSTP, HWRITEP, HSIZEP, HNONSECP, HEXCLP, HMASTERP, and HPROTP are checked when HTRANSP!=IDLE.</p> <p>HWDATAP is checked in data phase for write transfer.</p> <p>HRDATAP is checked in data phase for read transfer when HREADYP==1.</p> <p>HRESPP and HEXOKAYP are checked in data phase.</p>

Interface	Parity checking conditions
EPPB	<p>PSEL is always checked.</p> <p>PADDR, PPROT, PWRITE, PENABLE are checked when PSEL == 1.</p> <p>PREADY is checked when PSEL &amp;&amp; PENABLE.</p> <p>PWDATA and PSTRB are checked when PSEL &amp;&amp; PWRITE.</p> <p>PRDATA is checked when PSEL &amp;&amp; PENABLE &amp;&amp; PREADY &amp;&amp; !PWRITE.</p> <p>PSLVERR is checked when PSEL &amp;&amp; PENABLE &amp;&amp; PREADY.</p>
TCM-AHB	<p>HREADY, HREADYOUTS, HTRANS, HSELS, HADDRS, and SAHBWABORT are always checked.</p> <p>HBURSTS, HWrites, HSIZEs, HNONSECS, and HPROTS are checked when HTRANS != IDLE.</p> <p>HWDATAS and HWSTRBS are checked in data phase of NONSEQ and SEQ for write transfer.</p> <p>HRDATAS is checked in data phase for read transfer when HREADYOUTS ==1.</p> <p>HRESPS is checked in data phase.</p>
D-AHB	<p>HTRANS, HADDR, and HREADY are always checked.</p> <p>HBURST, HWRITE, HSIZE, HNONSEC, and HPROT are checked when HTRANS!=IDLE.</p> <p>HWDATAD is checked in data phase of NONSEQ and SEQ for write transfer.</p> <p>HRDATAD is checked in data phase for read transfer.</p> <p>HRESPD is checked in data phase.</p>
PMC-APB	<p>PMCPSEL is always checked.</p> <p>PMCPADDR, PMCPPROT, PMCPWRITE, PMCPENABLE are checked when PMCPSEL==1.</p> <p>PMCPREADY is checked when PMCPSEL &amp;&amp; PMCPENABLE.</p> <p>PMCPWDATA, PMCPSTRB are checked when PMCPSEL &amp;&amp; PMCPWRITE.</p> <p>PMCPRDATA is checked when PMCPSEL &amp;&amp; PMCPENABLE &amp;&amp; PMCPREADY &amp;&amp; !PMCPWRITE.</p> <p>PMCPSLVERR is checked when PMCPSEL &amp;&amp; PMCPENABLE &amp;&amp; PMCPREADY.</p>

Parity errors detected on the input signals on the interfaces are indicated to the system by a single-cycle pulse on one or more of the processor output signals, DBE. For more signal information, see [Error interface signals](#).

**Table 11-4: Detected parity error from the interface protection logic**

DBE[5:0]	Description
[5]	PMC-100 APB parity error
[4]	D-AHB parity error
[3]	M-AXI/M-AHB parity error



DBE[5:0]	Description
[2]	TCM-AHB parity error
[1]	P-AHB parity error
[0]	EPPB parity error

## 11.5 RAS memory barriers

The *Reliability, Availability, and Serviceability* (RAS) extension supports the *Error Synchronization Barrier* (ESB) instruction.

When this instruction is executed, all outstanding errors which have been detected but not reported are visible to the software running on the system. In the Cortex®-M52 processor, this instruction behaves in the same way as the *Data Synchronization Barrier* (DSB) instruction. When executed, all outstanding requests in the memory system are completed before the ESB instruction completes and any required BusFault exceptions are raised.

The RAS architecture supports another *Error Synchronization Barrier* (ESB) operation, which is implicit, that is, the *Implicit Error Synchronization Barrier* (IESB) operation. This feature is enabled by setting the AIRCR.IESB bit. When enabled, a barrier is inserted after the end of any register stacking or unstacking sequence associated with exception entry, exit, or floating-point register lazy stacking. Execution is halted in the processor until all outstanding transactions, including the stacking sequence have completed and any errors have been reported. The implicit barrier allows software to isolate an error during context switches, with RAS events always being reported in the old context.



Caution

Use IESB carefully because waiting for outstanding transactions to complete on exception entry can increase interrupt latency, particularly if an M-AXI/M-AHB access associated with the interrupted context takes many cycles to complete. The feature is disabled by default, with AIRCR.IESB set to 0 out of reset.

For more information on AIRCR, see the *Arm®v8-M Architecture Reference Manual*.

## 11.6 RAS Extension registers

The Cortex®-M52 processor implements the *Reliability, Availability, and Serviceability* (RAS) features to ensure correct operation in environments where functional safety and high-availability are critical. The RAS features can be controlled using the RAS Extension registers.

The following table lists the RAS Extension registers.

**Table 11-5: RAS Extension registers**

Address	Name	Type	Reset value	Description
0xE0005000	ERRFRO	RO	0x00000101 <b>Note:</b> 0x00000000, if the processor is not configured with <i>Error Correcting Code (ECC)</i> .	ERRFRO, RAS Error Record Feature Register
0xE0005008	ERRCTRL0	-	-	This register is <b>RES0</b> .
0xE0005010	ERRSTATUS0	RW	UNKNOWN	ERRSTATUS0, RAS Error Record Primary Status Register
0xE0005018	ERRADDR0	RW	UNKNOWN	ERRADDR0 and ERRADDR20, RAS Error Record Address Registers
0xE000501C	ERRADDR20	RO	UNKNOWN	ERRADDR0 and ERRADDR20, RAS Error Record Address Registers
0xE0005020	ERRMISC00	-	-	This register is <b>RES0</b> .
0xE0005024	ERRMISC10	RO	UNKNOWN	ERRMISC10, Error Record Miscellaneous Register 10
0xE0005028	ERRMISC20	-	-	This register is <b>RES0</b> .
0xE000502C	ERRMISC30	-	-	This register is <b>RES0</b> .
0xE0005030	ERRMISC40	-	-	This register is <b>RES0</b> .
0xE0005034	ERRMISC50	-	-	This register is <b>RES0</b> .
0xE0005038	ERRMISC60	-	-	This register is <b>RES0</b> .
0xE000503C	ERRMISC70	-	-	This register is <b>RES0</b> .
0xE0005E00	ERRGSRO	RO	0x00000000	ERRGSRO, RAS Fault Group Status Register
0xE0005FC8	ERRDEVID	RO	0x00000001 <b>Note:</b> 0x00000000, if the processor is not configured with ECC.	ERRDEVID, RAS Error Record Device ID Register
0xE000EF04	RFSR	RW	UNKNOWN	RFSR, RAS Fault Status Register

### 11.6.1 ERRFRO, RAS Error Record Feature Register

The *Reliability, Availability, and Serviceability* (RAS) ERRFRO register describes the RAS features that are supported.

#### Usage constraints

If the Security Extension is implemented and AIRCR.BFHFNMIN is zero, this register is RAZ/WI from the Non-secure state.

If the processor is not configured with ECC, this register is RAZ/WI.

Unprivileged access results in a BusFault exception.

#### Configurations

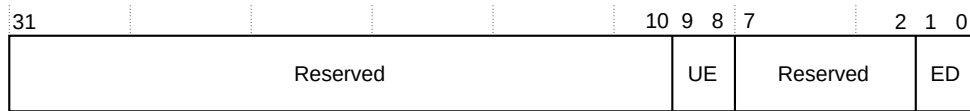
This register is always implemented.

## Attributes

This register is not banked between Security states. See [IMPLEMENTATION DEFINED registers summary](#) for more information.

The following figure shows the ERRFRO bit assignments.

**Figure 11-3: ERRFRO bit assignments**



The following table describes the ERRFRO bit assignments.

**Table 11-6: ERRFRO bit assignments**

Field	Name	Type	Description
[31:10]	Reserved	-	<b>RES0</b>
[9:8]	UE	RO	<p>Enable Uncorrected error (UE) reporting as an external abort.</p> <p><b>0b01</b> External abort response for uncorrected errors enabled.</p> <p>This field indicates that uncorrectable errors cause BusFault exceptions.</p>
[7:2]	Reserved	-	<b>RES0</b>
[1:0]	ED	RO	<p>Error reporting and logging.</p> <p><b>0b01</b> Reporting and logging always enabled.</p> <p>This field indicates that logging and reporting of errors cannot be disabled.</p>

## 11.6.2 ERRSTATUS0, RAS Error Record Primary Status Register

The Arm®v8.1-M *Reliability, Availability, and Serviceability* (RAS) ERRSTATUS0 register contains information about the *Reliability, Availability, and Serviceability* (RAS) event that is currently logged in record 0.

### Usage constraints

If the Security Extension is implemented and AIRCR.BFHFNMINS is zero, this register is RAZ/WI from the Non-secure state.

If the processor is not configured with ECC, this register is RAZ/WI.

Unprivileged access results in a BusFault exception.

### Configurations

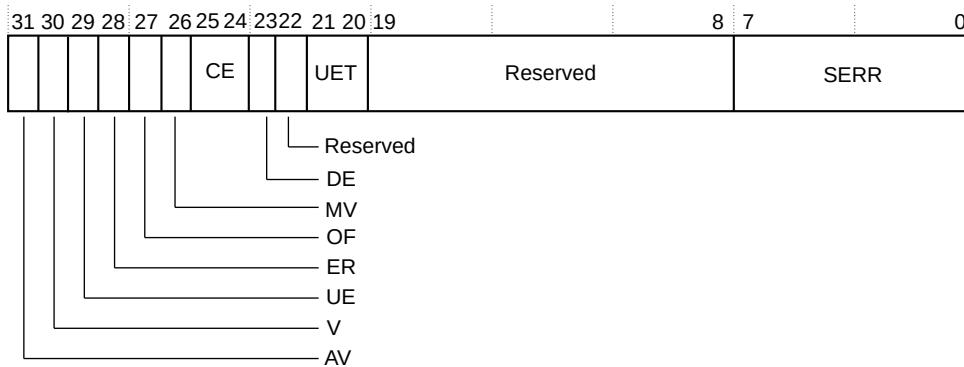
This register is always implemented.

## Attributes

The register is not banked between Security states. The read/write behavior depends on the individual fields. See [IMPLEMENTATION DEFINED registers summary](#) for more information.

The following figure shows the ERRSTATUS0 bit assignments.

**Figure 11-4: ERRSTATUS0 bit assignments**



The following table describes the ERRSTATUS0 bit assignments.

**Table 11-7: ERRSTATUS0 bit assignments**

Field	Name	Type	Description
[31]	AV	RW	<p>Address valid.</p> <p><b>0b0</b> ERRADDR0 is not valid. <b>0b1</b> ERRADDR0 is valid.</p> <p>ERRADDR0 is valid only if:</p> <ul style="list-style-type: none"> <li>A precise BusFault caused the RAS event.</li> <li>A TCM Error Correcting Code (ECC) error caused the RAS event.</li> </ul> <p>This bit is write-one-to-clear.</p>
[30]	V	RW	<p>Status valid.</p> <p><b>0b0</b> ERRSTATUS0 is not valid. <b>0b1</b> ERRSTATUS0 is valid.</p> <p>This field is set to 1 on any RAS event.</p>
[29]	UE	RW	<p>Uncorrected errors (UEs).</p> <p><b>0b0</b> No uncorrectable errors detected. <b>0b1</b> At least one uncorrectable error is detected.</p> <p>This bit is write-one-to-clear.</p>

Field	Name	Type	Description
[28]	ER	RW	<p>Error reported.</p> <p><b>0b0</b> No BusFault caused by RAS event has occurred. <b>0b1</b> BusFault caused by RAS event has occurred.</p> <p>This bit is write-one-to-clear.</p>
[27]	OF	RW	<p>Overflow.</p> <p><b>0b0</b> At most one RAS event has occurred since the last time ERRSTATUS.V was cleared. <b>0b1</b> At least two RAS events have occurred since the last time ERRSTATUS.V was cleared. These events might have occurred at the same time.</p> <p>This bit is write-one-to-clear.</p>
[26]	MV	RW	<p>Miscellaneous registers valid.</p> <p><b>0b0</b> ERRMISCO is not valid. <b>0b1</b> ERRMISCO is valid.</p> <p>This field is set to 1 on any RAS event. This bit is write-one-to-clear.</p>
[25:24]	CE	RW	<p>Corrected errors.</p> <p><b>0b00</b> Corrected errors (CEs) have not been detected. <b>0b10</b> At least one Corrected error (CE) has been detected.</p> <p>This bit is write-one-to-clear.</p>
[23]	DE	RW	<p>Deferred errors.</p> <p><b>0b0</b> No errors were deferred. <b>0b1</b> At least one error was deferred.</p> <p>This bit is write-one-to-clear.</p>
[22]	PN	-	<p>Poison.</p> <p><b>0b0</b> No BusFault due to TEBRx.POSON has occurred. <b>0b1</b> At least one BusFault due to TEBRx.POSON has occurred.</p>
[21:20]	UET	RW	<p>Uncorrectable error type.</p> <p><b>0b00</b> Uncorrectable error, Uncontainable error (UC). This is for any uncorrectable error that caused an asynchronous BusFault <b>0b11</b> Uncorrectable error, Recoverable error (UER). This is for an uncorrectable error that caused a synchronous BusFault</p> <p>These bits are write-one-to-clear (0b11)</p>
[19:8]	Reserved	-	<b>RES0</b>

Field	Name	Type	Description
[7:0]	SERR	RW	<p>Architecturally-defined primary error code.</p> <p><b>0</b> No error.</p> <p><b>2</b> TCM ECC error.</p> <p><b>6</b> L1 data cache or instruction cache or unified cache data RAM ECC error.</p> <p><b>7</b> L1 data cache or instruction cache or unified cache tag RAM ECC error.</p> <p><b>21</b> Poison BusFault due to TEBRx.POISON.</p> <p>The Cortex®-M52 processor does not use the other values of this field.</p>

### 11.6.3 ERRADDR0 and ERRADDR20, RAS Error Record Address Registers

The *Reliability, Availability, and Serviceability* (RAS) ERRADDR0 and ERRADDR20 registers contain information about the address of the *Reliability, Availability, and Serviceability* (RAS) event in record 0.

#### Usage constraints

If the Security Extension is implemented and AIRCR.BFHFNMINS is zero, this register is RAZ/WI from the Non-secure state.

If the processor is not configured with ECC, this register is RAZ/WI.

Unprivileged access results in a BusFault exception.

This register ignores writes if ERRSTATUS0.AV is set to 1.

#### Configurations

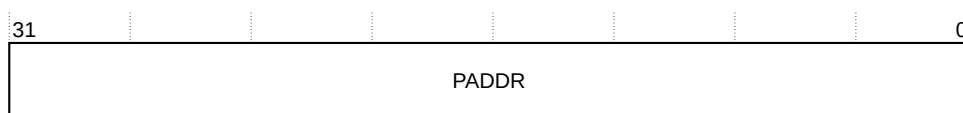
These registers are always implemented.

#### Attributes

These registers are not banked between Security states. See [IMPLEMENTATION DEFINED registers summary](#) for more information.

The following figure shows the ERRADDR0 bit assignments.

**Figure 11-5: ERRADDR0 bit assignments**



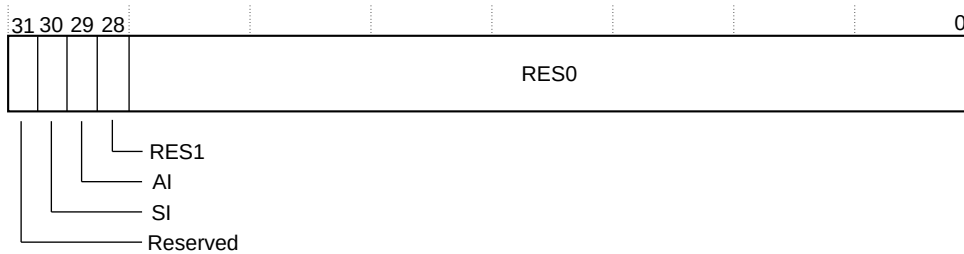
The following table describes the ERRADDR0 bit assignments.

**Table 11-8: ERRADDR0 bit assignments**

Field	Name	Type	Description
[31:0]	PADDR	RW	Address of the RAS event. This is the address associated with the memory access that observed <i>Error Correcting Code</i> (ECC) error. This field is not valid if ERRADDR20.AI is 0b1.

The following figure shows the ERRADDR20 bit assignments.

**Figure 11-6: ERRADDR20 bit assignments**



The following table describes the ERRADDR20 bit assignments.

**Table 11-9: ERRADDR20 bit assignments**

Field	Name	Type	Description
[31]	Reserved	-	<b>RES0</b>
[30]	SI	RO	Security information incorrect.  <b>0b1</b> NS bit is not valid.  The security information is never guaranteed to be correct.
[29]	AI	RO	Address incorrect.  <b>0b0</b> PADDR is valid. <b>0b1</b> PADDR is not valid.  PADDR is valid only if: <ul style="list-style-type: none"> <li>The RAS event was a precise BusFault.</li> <li>The RAS event was associated with a TCM ECC error.</li> </ul> <b>Note:</b> If software clears ERRSTATUS.AV, then ERRADDR20.AI is set to 0b1 to invalidate the address.
[28]	Reserved	-	<b>RES1</b>
[27:0]	Reserved	-	<b>RES0</b>

## 11.6.4 ERRMISC10, Error Record Miscellaneous Register 10

The ERRMISC10 register is an **IMPLEMENTATION DEFINED** error syndrome register for the event in record 0.

### Usage constraints

If the Security Extension is implemented and AIRCR.BFHFNMIN is zero, this register is RAZ/WI from the Non-secure state.

If the processor is not configured with *Error Correcting Code* (ECC), this register is RAZ/WI. Unprivileged access results in a BusFault exception.

### Configurations

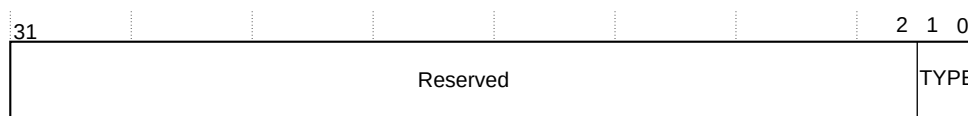
This register is always implemented.

### Attributes

This register is not banked between Security states. See [IMPLEMENTATION DEFINED registers summary](#) for more information.

The following figure shows the ERRMISC10 bit assignments.

**Figure 11-7: ERRMISC10 bit assignments**



The following table describes the ERRMISC10 bit assignments.

**Table 11-10: ERRMISC10 bit assignments**

Field	Name	Type	Description
[31:2]	Reserved	-	<b>RES0</b>
[1:0]	TYPE	RO	Indicates the type of <i>Reliability, Availability, and Serviceability</i> (RAS) event logged.  <b>0b00</b> L1 instruction cache ECC. <b>0b01</b> L1 data cache or unified cache ECC. <b>0b10</b> TCM ECC found by load or store executed by the processor. <b>0b11</b> TCM ECC found by access from <i>AHB TCM Access</i> (TCM-AHB).



In the Cortex®-M52 processor, only ERRMISC10 is implemented. ERRMISC00 and ERRMISC20-ERRMISC70 are **RES0**.

## 11.6.5 ERRGSR0, RAS Fault Group Status Register

The ERRGSR0 register summarizes the valid error records. The Cortex®-M52 processor only supports one error record, therefore, only one bit of ERRGSR is active.

### Usage constraints

If the Security Extension is implemented and AIRCR.BFHFNMINS is zero, this register is RAZ/WI from the Non-secure state.



If the processor is not configured with *Error Correcting Code* (ECC), this register is RAZ/WI. Unprivileged access results in a BusFault exception.

Configurations

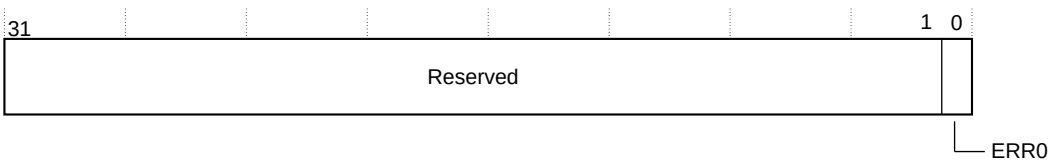
This register is always implemented.

Attributes

This register is not banked between Security states. See [IMPLEMENTATION DEFINED registers summary](#) for more information.

The following figure shows the ERRGSRO bit assignments.

Figure 11-8: ERRGSRO bit assignments



The following table describes the ERRGSRO bit assignments.

Table 11-11: ERRGSRO bit assignments

Field	Name	Type	Description
[31:1]	Reserved	-	RES0
[0]	ERR0	RO	Error record 0 is valid.

11.6.6 ERRDEVID, RAS Error Record Device ID Register

The *Reliability, Availability, and Serviceability* (RAS) ERRDEVID register contains the number of error records that an implementation supports. The Cortex®-M52 processor supports a single error record with index 0 if *Error Correcting Code* (ECC) is configured or there are no error records.

Usage constraints

Unprivileged access results in a BusFault exception.  
This register is accessible through unprivileged *Debug AHB* (D-AHB) debug requests when either DAUTHCTRL\_S.UIDAPEN or DAUTHCTRL\_NS.UIDAPEN is set.

Configurations

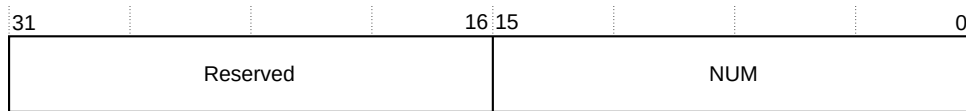
This register is always implemented.

Attributes

This register is not banked between Security states. See [IMPLEMENTATION DEFINED registers summary](#) for more information.

The following figure shows the ERRDEVID bit assignments.

**Figure 11-9: ERRDEVID bit assignments**



The following table describes the ERRDEVID bit assignments.

**Table 11-12: ERRDEVID bit assignments**

Field	Name	Type	Description
[31:16]	Reserved	-	<b>RES0</b>
[15:0]	NUM	RO	<p>Maximum Error Record Index+1</p> <p><b>0x0001</b> If ECC is configured, then one error record with index 0.  <b>0x0000</b> If ECC is not configured, then there are no error record registers.</p> <p><b>Note:</b></p> <ul style="list-style-type: none"> <li>ECC is configured using the Verilog parameter <code>ECC</code> and enabled by driving the input signal <code>INITECCEN</code> to 1.</li> <li>ERRDEVID[0] always reads the same value as <code>MSCR.ECCEN</code>.</li> </ul>

### 11.6.7 RFSR, RAS Fault Status Register

The RFSR reports the fault status of *Reliability, Availability, and Serviceability* (RAS) related faults from *Error Correcting Code* (ECC) errors that are detected in the L1 instruction cache, unified cache, data cache, and TCM.

#### Usage constraints

If the Security Extension is implemented and `AIRCR.BFHFNMINS` is zero, this register is RAZ/WI from Non-secure state.

If the processor is not configured with *Error Correcting Code* (ECC), this register is RAZ/WI. Unprivileged access results in a BusFault exception.

#### Configurations

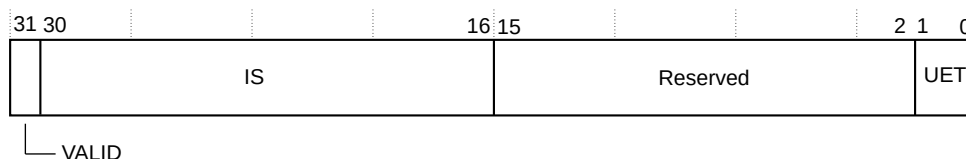
This register is always implemented.

#### Attributes

This register are not banked between Security states. See [IMPLEMENTATION DEFINED registers summary](#) for more information.

The following figure shows the RFSR bit assignments.

**Figure 11-10: RFSR bit assignments**



The following table describes the RFSR bit assignments.

**Table 11-13: RFSR bit assignments**

Bit	Name	Type	Description
[31]	Valid	RW	Indicates whether the register is valid. This bit is write-one-to-clear and therefore, it is cleared by writing 1. Writes of zero are ignored.
[30:16]	IS	RW	<b>IMPLEMENTATION-DEFINED</b> syndrome. Indicates the type of RAS exception that has occurred. <ul style="list-style-type: none"> <li><b>0x0</b> L1 instruction cache ECC.</li> <li><b>0x1</b> L1 data cache or unified cache ECC.</li> <li><b>0x2</b> TCM ECC.</li> </ul>
[15:2]	Reserved	-	<b>RES0.</b>
[1:0]	UET	RW	Error type. <ul style="list-style-type: none"> <li><b>0b00</b> Uncontainable error (UC). RAS exception is imprecise.</li> <li><b>0b11</b> Recoverable error (UER). RAS exception is precise.</li> </ul> For more information on error types, see the <a href="#">ECC schemes and error type terminology</a> .

# 12. Nested Vectored Interrupt Controller

This chapter describes the *Nested Vectored Interrupt Controller* (NVIC).

## 12.1 NVIC features

The Cortex®-M52 processor *Nested Vectored Interrupt Controller* (NVIC) is closely integrated with the core to achieve low-latency interrupt processing.

The NVIC is responsible for:

- Maintaining the current execution priority of the Cortex®-M52 processor.
- Maintaining the pending and active status of all exceptions that are supported.
- Invoking preemption when a pending exception has priority.
- Providing wakeup signals to wakeup the Cortex®-M52 processor from deep sleep mode.
- Providing support to the *Internal Wakeup Interrupt Controller* (IWIC) and *External Wakeup Interrupt Controller* (EWIC).
- Providing priority and exception information to other processor components.

The NVIC in the Cortex®-M52 processor allows up to 496 exceptions, of which, 480 can be regular external interrupts.

## 12.2 Registers associated with interrupt control and behavior

Registers associated with interrupt control and interrupt behavior are found in the following categories.

**Table 12-1: Interrupt control and behavior registers**

Register summary	Registers	Description
System control block	<ul style="list-style-type: none"> <li>• ICSR</li> <li>• AIRCR</li> <li>• SHPR1-3</li> </ul>	<a href="#">System control register summary</a>
Implementation control block	ICTR	<a href="#">Implementation control register summary</a>
Software Interrupt Generation	STIR	<a href="#">Software Interrupt Generation register summary</a>
SysTick Timer	<ul style="list-style-type: none"> <li>• SYST_CSR</li> <li>• SYST_RVR</li> <li>• SYST_CVR</li> <li>• SYST_CALIB</li> </ul>	<a href="#">SysTick Timer register summary</a>

## 12.3 NVIC register summary

The *Nested Vectored Interrupt Controller* (NVIC) registers can be accessed through the *Internal Private Peripheral Bus* (IPPB) interface. Each of the NVIC registers is 32 bits wide.

The NVIC\_ISERn, NVIC\_ICERn, NVIC\_ISPRn, NVIC\_ICPRn, NVIC\_IABRn, and NVIC\_IPRn registers are not banked between Security states. If an interrupt is configured as Secure in the NVIC\_ITNSn register, any access to the corresponding NVIC\_ISERn, NVIC\_ICERn, NVIC\_ISPRn, NVIC\_ICPRn, NVIC\_IABRn, or NVIC\_IPRn registers from Non-secure are treated as RAZ/WI.

For more information on the NVIC registers listed in the following table, see *Arm®v8-M Architecture Reference Manual*.

**Table 12-2: NVIC register summary**

Address offset	Name	Type	Reset value	Description
0xE000E100 - 0xE000E13C	NVIC_ISER0- NVIC_ISER15	RW	0x00000000	Interrupt Set-Enable Registers
0xE000E180 - 0xE000E1BC	NVIC_ICER0- NVIC_ICER15	RW	0x00000000	Interrupt Clear-Enable Registers
0xE000E200 - 0xE000E23C	NVIC_ISPR0- NVIC_ISPR15	RW	0x00000000	Interrupt Set-Pending Registers
0xE000E280 - 0xE000E2BC	NVIC_ICPR0- NVIC_ICPR15	RW	0x00000000	Interrupt Clear-Pending Registers
0xE000E300 - 0xE000E33C	NVIC_IABR0- NVIC_IABR15	RO	0x00000000	Interrupt Active Bit Register
0xE000E380 - 0xE000E3BC	NVIC_ITNS0- NVIC_ITNS15	RW	0x00000000	Interrupt Target Non-secure Registers <b>Note:</b> These registers are Secure only. They are RAZ/WI when accessed from Non-secure state.
0xE000E400 - 0xE000E5DC	NVIC_IPRO- NVIC_IPR119	RW	0x00000000	Interrupt Priority Registers

## 12.4 Software Interrupt Generation register summary

The following table shows the architecturally defined Software Interrupt Generation register.

**Table 12-3: Software Interrupt Generation register summary**

Address offset	Name	Type	Reset value	Description
0xE000EF00	STIR	WO	0x00000000	Software Triggered Interrupt Register. For more information, see <i>Arm®v8-M Architecture Reference Manual</i> .

## 12.5 SysTick Timer register summary

The following table shows the architecturally defined SysTick Timer registers.



For more information on the architectural registers listed in the following table, see the *Arm®v8-M Architecture Reference Manual*.

**Table 12-4: SysTick Timer register summary**

Address offset	Name	Type	Reset value	Description
0xE000E010	SYST_CSR	RW	0x00000000	SysTick Control and Status Register
0xE000E014	SYST_RVR	RW	0x00000000	SysTick Reload Value Register
0xE000E018	SYST_CVR	RW	0x00000000	SysTick Current Value Register
0xE000E01C	SYST_CALIB	RO	0x00000000	SysTick Calibration Value Register

## 13. External coprocessors

This chapter describes the interface and programmer's model for connecting and using external coprocessors.

### 13.1 External coprocessors features

The Cortex®-M52 processor supports an external coprocessor interface which allows the integration of tightly coupled accelerator hardware with the processor. The programmers model allows software to communicate with the hardware by using architectural coprocessor instructions.

The external coprocessor interface:

- Supports low-latency data transfer from the processor to and from the accelerator components.
- Provides a mechanism for you to extend the capabilities of the Cortex®-M52 processor.
- Supports up to eight separate coprocessors, CP0-CP7, depending on your implementation. The remaining coprocessor numbers, CP8-CP15, are reserved. CP10 and CP11 are always reserved for floating-point or *M-profile Vector Extension* (MVE) functionality. For more information, see the *Arm®v8-M Architecture Reference Manual*. The Cortex®-M52 processor system can configure which coprocessor is included in Secure and Non-secure states.

For each coprocessor CP0-CP7, the encoding space can be dedicated to either the external coprocessor or the *Custom Datapath Extension* (CDE) modules. See [Arm Custom Instructions](#) for information on the CDE implementation in the processor.

### 13.2 Operation

The external coprocessor interface provides control and data channels for up to eight separate coprocessors. The external devices are provided with information about privilege and Security state of the processor with the instruction type and associate register and operation fields that the architecture defines. The following instruction types are supported:

- Register transfer from the Cortex®-M52 processor to the coprocessor MCR, MCRR, MCR2, MCRR2.
- Register transfer from the coprocessor to the Cortex®-M52 processor MRC, MRRC, MRC2, MRRC2.
- Data processing instructions CDP, CDP2.

The interface provides a handshake mechanism to indicate to the coprocessor that an instruction has been committed in the processor and can no longer be interrupted. Additionally, it can stall the processor in a way that it can always be interrupted (BUSYWAIT) and to indicate that an error has occurred while waiting for an UNDEFINSTR UsageFault.



Note

- The regular and extension forms of the coprocessor instructions for example, `MCR` and `MCRR2`, have the same functionality but different encodings. The two encoding values differ by a single bit, bit [12]. For more information, see the *Arm®v8-M Architecture Reference Manual*.
- The `MRC` and `MRC2` instructions support the transfer of APSR.NZVC flags when the processor register field is set to PC, for example `Rt == 0xF`.

## 13.3 Data transfer rates

The following table lists the ideal data transfer rates for the coprocessor interface. This means that the coprocessor responds to an instruction immediately and does not `BUSYWAIT`. The ideal data transfer rates are sustainable if the corresponding coprocessor instructions are executed consecutively.

**Table 13-1: Ideal data transfer rates for the coprocessor interface**

Instructions	Direction	Ideal data rate
<code>MCR</code> , <code>MCR2</code>	Processor to coprocessor	32 bits per cycle
<code>MRC</code> , <code>MRC2</code>	Coprocessor to processor	32 bits per cycle
<code>MCRR</code> , <code>MCRR2</code>	Processor to coprocessor	64 bits per cycle
<code>MRRC</code> , <code>MRRC2</code>	Coprocessor to processor	64 bits per cycle

## 13.4 Coprocessor instruction restrictions

The following restrictions apply when the Cortex®-M52 processor uses coprocessor instructions:

- The `LDC (2)` or `STC (2)` instructions are not supported. If these are included in software with the `<coproc>` field set to a value between 0-7 and the coprocessor is present and enabled in the appropriate fields in the CPACR or NSACR, the Cortex®-M52 processor always attempts to take an *Undefined instruction* (UNDEFINSTR) UsageFault exception.
- The processor register fields for data transfer instructions must not include the stack pointer (`Rt = 0xD`), this encoding is **UNPREDICTABLE** in the Arm®v8.1-M architecture and results in an UNDEFINSTR UsageFault exception in the Cortex®-M52 processor if the coprocessor is present and enabled in the CPACR or NSACR.
- If any coprocessor instruction is executed when the corresponding coprocessor is either not present or disabled in the CPACR or NSACR, the Cortex®-M52 processor always attempts to take a *No coprocessor* (NOCP) UsageFault exception.

For more information on the CPACR and NSACR, see the *Arm®v8-M Architecture Reference Manual*.



## 13.5 Debug access to coprocessor registers usage constraints

The Cortex®-M52 processor does not support a mechanism to read and write registers located in external coprocessors.

Arm® recommends that you implement a coprocessor with a dedicated AHB or APB slave interface for the system to access the registers. If the debug view of the coprocessor is located in the PPB region of the memory map, you can use this interface to connect to the *External Private Peripheral Bus* (EPPB) interface of the Cortex®-M52 processor.

If Secure debug is disabled, you must ensure the Secure information in the coprocessors is protected and not accessible when using a Non-secure debugger.

If the debug slave interface to the coprocessor is connected to the processor Main interface or *Peripheral AHB* (P-AHB) interfaces or the EPPB interface, you can use the ARPROT[1], AWPROT[1], HNONSECC, HNONSECSYS, HNONSECP, and PPROT[2] signals on the M-AXI (or M-AHB), P-AHB, and APB interfaces respectively. This is because the security level of the debug requests routed through the processor from the D-AHB interface are subject to the debug access and authentication checks.

If the coprocessor state is memory-mapped, then software can also access the information using load and store instructions. If your implementation uses this functionality, you must ensure the appropriate barrier instructions are included to guarantee ordering between coprocessor instructions and load/store operations to the same state.

## 13.6 Exceptions and context switch

The Cortex®-M52 processor does not include support for automatic save and restore of coprocessor registers on entry and exit to exceptions, unlike the internal processor integer and floating-point registers. Any coprocessor state that must be maintained across a context switch must be carried out by the software that is aware of the coprocessor requirements.

You must ensure that when the coprocessor contains Secure data, it cannot be accessed by software running in a Non-secure exception handler.

## 13.7 Response to coprocessor errors

The coprocessor must not rely on a synchronous exception that is taken when asserting a CPERROR response to a coprocessor transaction, because the UNDEFINSTR UsageFault might be preempted by a higher priority interrupt in the Cortex®-M52 processor. There is no guarantee that there are no side effects from the erroneous instruction.

## 13.8 Hazard between load and store instructions followed by coprocessor transactions

A possible hazard exists when a load store instruction is followed by coprocessor transactions.

To decouple the data side TCMWAIT input signal from the CPVALID output signal, a coprocessor instruction following a load or store instruction in the processor always stalls for a clock cycle after the load or store completes.

This situation does not add stall cycles to the data hazard that is already included a coprocessor data transfer instruction consumes the result of a load. That is the most common case of data hazard.

# 14. Arm Custom Instructions

This chapter describes the support for *Arm Custom Instructions* (ACI) and the implementation of the *Custom Datapath Extension* (CDE) in the processor.

## 14.1 Arm Custom Instructions support

The Cortex®-M52 processor supports *Arm Custom Instructions* (ACIs) and implements the *Custom Datapath Extension* (CDE) for Arm®v8-M.

The ACI support provides the following:

- New architecturally defined instructions.
- Interfaces that support the addition of user-defined instructions.
- Compliance tests to check the integration of the user-defined instructions as part of the execution testbench.

### CDE modules

For each coprocessor CP0-CP7, the CDE architecture allows you to choose to either use the external coprocessor interface or bypass it and use CDE modules instead.

The Cortex®-M52 processor includes a core CDE module and a floating-point and MVE CDE module.

You are responsible for the content of these modules in your implementation. Arm is responsible for the interfaces to these modules.

### CDE

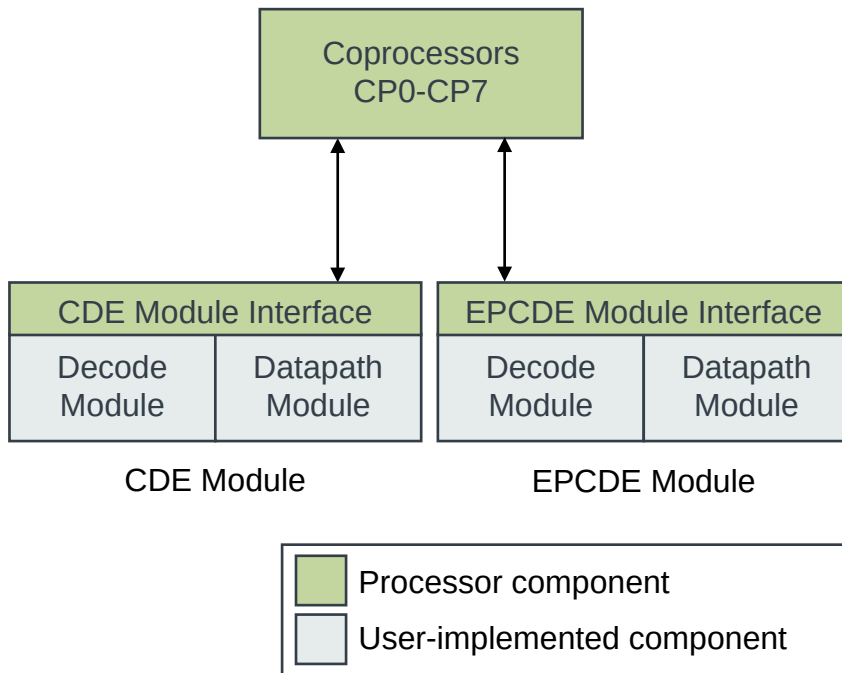
The core CDE module executes instructions that access the general-purpose registers and the APSR register. This module is reset and clocked in the same way as the processor core, and it is included in the Core power domain.

### EPCDE

The floating-point and MVE CDE module executes instructions that access the floating-point and MVE registers. This module is reset and clocked in the same way as the *Extension Processing Unit* (EPU). If the core CDE module is present and used, and if the EPU is present, then the floating-point and MVE CDE module is also present.

The CDE and EPCDE modules both have a decode and datapath module. The following figure shows the coprocessors CP0-CP7 connected to the CDE and EPCDE modules and interfaces.

### Figure 14-1: CDE and EPCDE module interfaces



## User-defined instructions

The CDE architecture defines instruction classes depending on the number of source or destination registers. For each class, an accumulation variant exists. You define the function of these instruction classes in the dedicated CDE module added to the processor core or to the EPU.

The classes are:

CX1, CX2, CX3

These three classes operate on the general-purpose register file, including the condition flags APSR nzc.

You can define different functions for a given instruction class depending on the coprocessor number and the opcode value `<imm>`.

## VCX1, VCX2, VCX3

These three classes operate on the floating-point register file only.

You can define different functions for a given instruction class depending on the coprocessor number and the opcode value `<imm>`.

## VCX1 (Vector), VCX2 (Vector), VCX3 (Vector)

These three classes operate on the MVE file only.

You can define different functions for a given instruction class depending on the coprocessor number and the opcode value `<imm>`.

## ACI support in multi-Cortex®-M52 systems with different CDE customization

In a system with several Cortex®-M52 processors, it is possible to configure a different CDE customization for each processor using the `CDERTLID` parameter. This parameter can be used to implement different functions for an identical instruction.

Software can read the `CDERTLID` parameter using any of the following registers:

**Table 14-1: Cortex®-M52 registers that can read the `CDERTLID` parameter**

Name	Description
ID_AFR0	ID_AFR0, Auxiliary Feature Register 0
CFGINFOSEL	CFGINFOSEL, Processor configuration information selection register
CFGINFORD	CFGINFORD, Processor configuration information read data register

## 14.2 Usage restrictions

Some restrictions apply when the Cortex®-M52 processor uses *Custom Datapath Extension* (CDE) instructions.

Depending on your processor implementation at hardware and software level and on your implementation of the CDE and EPCDE modules, NOCP or UNDEFINSTR exceptions might occur when *Arm Custom Instructions* (ACIs) are in use.

For more information on usage restrictions and fault behavior see *Exceptions in the CDE and EPCDE modules* section in the *Cortex®-M52 Processor Integration and Implementation Manual*. The *Cortex®-M52 Processor Integration and Implementation Manual* is a confidential document that is only available to licensees and Arm partners with an NDA agreement.

# 15. Floating-point and MVE support

This chapter describes the *Extension Processing Unit* (EPU), which controls floating-point and *M-profile Vector Extension* (MVE) support.

## 15.1 Floating-point and MVE operation

The *Extension Processing Unit* (EPU) can be configured to perform floating-point and *M-profile Vector Extension* (MVE) operations.

### Scalar floating-point operation

The Cortex®-M52 processor can be configured to provide scalar half, single, and double-precision floating-point operation. The floating-point operation is an implementation of the scalar half, single, and double-precision variants of the Floating-point Extension, FPUv5 architecture. Configuring the processor to include floating-point supports all half, single, and double-precision data-processing instructions and data types described in the *Arm®v8-M Architecture Reference Manual*.

The processor supports scalar half, single, and double-precision add, subtract, multiply, divide, multiply and accumulate, and square root operations. The floating-point functionality that the processor supports also provides conversions between fixed-point and floating-point data formats, and floating-point constant instructions.

### M-profile Vector Extension operation

The Cortex®-M52 processor can be configured to provide MVE operation. The MVE functionality that is supported depends on the inclusion of floating-point functionality.

- If floating-point functionality is not included, the processor can be configured to any of the following:
  - Not include MVE.
  - Include the integer subset of MVE only (MVE-I). MVE-I operates on 8-bit, 16-bit, and 32-bit data types.
- If floating-point functionality is included, the processor can be configured to any of the following:
  - Not include MVE. Include scalar half-precision and single-precision floating-point.
  - Not include MVE. Include scalar half-precision, single-precision, and double-precision floating-point.
  - Include the integer subset of MVE only (MVE-I), scalar half-precision and single-precision floating-point. MVE-I operates on 8-bit, 16-bit, and 32-bit data types.
  - Include the integer MVE (MVE-I), half-precision, and single-precision floating-point MVE (MVE-F), scalar half-precision, single-precision, and double-precision floating-point. MVE-F operates on half-precision and single-precision floating-point values.

Vector instructions operate on a fixed vector width of 128 bits. The lane width of an operation to be performed is specified by the instruction that is being executed. And an

element refers to the data that is put into a lane. Multiple lanes can be executed per beat. There are four beats per vector instruction.

For more information on the MVE extension and terminology, see *Arm®v8-M Architecture Reference Manual*.



Note

- The Cortex®-M52 processor provides floating-point computation functionality included with the MVE and Floating-point Extension, which is compliant with the *ANSI/IEEE Std 754-2008, IEEE Standard for Binary Floating-Point Arithmetic*.
- The scalar Floating-point Extension can be implemented with or without *M-profile Vector Extension - floating-point (MVE-F)*.

### 15.1.1 EPU views of the register bank

The *Extension Processing Unit (EPU)* provides an extension register file with registers that can be viewed as:

- Thirty-two 32-bit single-word registers, S0-S31.
- Sixteen 64-bit doubleword registers, D0-D15.
- Eight 128-bit vector registers, Q0-Q7.
- A combination of registers from these views.

### 15.1.2 Modes of operation

The Cortex®-M52 processor supports the following modes of operation:

- Flush to-zero
- Half-precision flush to-zero
- Default NaN

For more information on these modes, see the *Arm®v8-M Architecture Reference Manual*.

### 15.1.3 Compliance with the IEEE 754 standard

The Cortex®-M52 processor provides floating-point computation functionality included with the MVE and Floating-point Extension, which is compliant with the *ANSI/IEEE Std 754-2008, IEEE Standard for Binary Floating-Point Arithmetic*. No support code is required to achieve this compliance.

## 15.1.4 Exceptions

The *Extension Processing Unit* (EPU) sets the cumulative exception status flags in the FPSCR register as required for each instruction, in accordance with the Fpv5 architecture. The EPU does not support exception traps.

The processor also has six output pins, each pin reflects the status of one of the cumulative exception flags:

- Inexact result.
- The input is denormal.
- Overflow.
- Underflow.
- Divide-by-zero.
- Invalid operation

## 15.2 Floating-point and MVE register summary

The *Extension Processing Unit* (EPU) has various registers that support floating-point and *M-profile Vector Extension* (MVE) operations.

The following table shows a summary of the floating-point registers. These registers are described in the *Arm®v8-M Architecture Reference Manual*.



FPCCR, FPCAR, and FPDSCR are banked between Security states.

**Table 15-1: Floating-point and MVE register summary**

Address	Name	Type	Reset value	Description
0xE000EF34	FPCCR	RW	0xC0000004	Floating-point Context Control Register (S)
0xE000EF38	FPCAR	RW	0x00000000	Floating-point Context Address Register (S)
0xE000EF3C	FPDSCR	RW	See <a href="#">FPDSCR</a> and <a href="#">FPSCR</a> register reset values	Floating-point Default Status Control Register (S)
This register is not memory mapped	FPSCR	RW		Floating-point Status and Control Register
0xE000EF40	MVFR0	RO	<a href="#">MVFR0</a> , <a href="#">MVFR1</a> , and <a href="#">MVFR2</a> reset values	Media and VFP Feature Register 0
0xE000EF44	MVFR1	RO		Media and VFP Feature Register 1
0xE000EF48	MVFR2	RO		Media and VFP Feature Register 2



## 15.3 FPDSCR and FPSCR register reset values

The following table shows the reset values for *Floating-point Default Status Control Register* (FPDSCR) and *Floating-point Status and Control Register* (FPSCR) depending on inclusion and exclusion of floating-point and *M-profile Vector Extension* (MVE) functionality.

**Table 15-2: FPDSCR and FPSCR reset values**

Register name	Reset value	Floating-point and MVE configuration
FPDSCR	0x00000000	Floating-point and MVE are not included.
	0x00040000	Scalar half and single-precision floating-point is included. MVE is not included.
		Scalar half, single, and double-precision floating-point is included. MVE is not included.
		Floating-point is not included. Integer subset of MVE is included.
		Scalar half and single-precision floating-point is included. Integer subset of MVE is included.
		Scalar half, single, and double-precision floating-point is included. Integer and half and single-precision floating-point MVE is included.
FPSCR	RES0	Floating-point and MVE are not included.
	UNKNOWN	Scalar half and single-precision floating-point is included. MVE is not included.
		Scalar half, single, and double-precision floating-point is included. MVE is not included.
		Floating-point is not included. Integer subset of MVE is included.
		Scalar half and single-precision floating-point is included. Integer subset of MVE is included.
		Scalar half, single, and double-precision floating-point is included. Integer and half and single-precision floating-point MVE is included.

# 16. Debug

This chapter describes the debug system.

## 16.1 Debug functionality

The Cortex®-M52 processor debug functionality includes Arm®v8-M, Arm®v8.1-M, and CoreSight™ features that are designed to support debug and trace of software running on the processor.

These features include:

- A *BreakPoint Unit* (BPU) which can be configured to support four or eight hardware breakpoints.
- A *Data Watchpoint and Trace* (DWT) unit which can be configured to support two, four, or eight hardware comparators that can match both address and data values.
- Support for the *Digital Signal Processing* (DSP) debug extension for analysis of signal processing and compute-based software.
- Monitor mode exception for self-hosted debug.
- Full access to the memory map and registers through a 32-bit *Debug AHB* (D-AHB) interface.
- An *Instrumentation Trace Macrocell* (ITM) for software-driven `printf` debugging which can be linked to the DWT.
- An implementation of the *Performance Monitoring Unit* (PMU).
- An *Embedded Trace Macrocell* (ETM) which supports complete instruction trace. It implements the ETMv4.5 architecture, including support for tracing the *M-profile Vector Extension* (MVE) features. Data trace is not supported. For more information on the ETM, see the *Arm China CoreSight™ ETM-M52 Technical Reference Manual*.
- Access control that prevents unauthorized debug or trace of Secure state or memory, including support for the Unprivileged Debug Extension for fine-grain control of debug access to the processor.



- Except for debug monitor mode, all other debug and trace functionality on the Cortex®-M52 processor is optional.
  - The debugger cannot write to the *Interrupt Program Status Register* (IPSR).
  - The Cortex®-M52 processor is also supplied with an optional *Trace Port Interface Unit* (TPIU). For more information, see [Trace Port Interface Unit](#).
-

### 16.1.1 CoreSight™ discovery

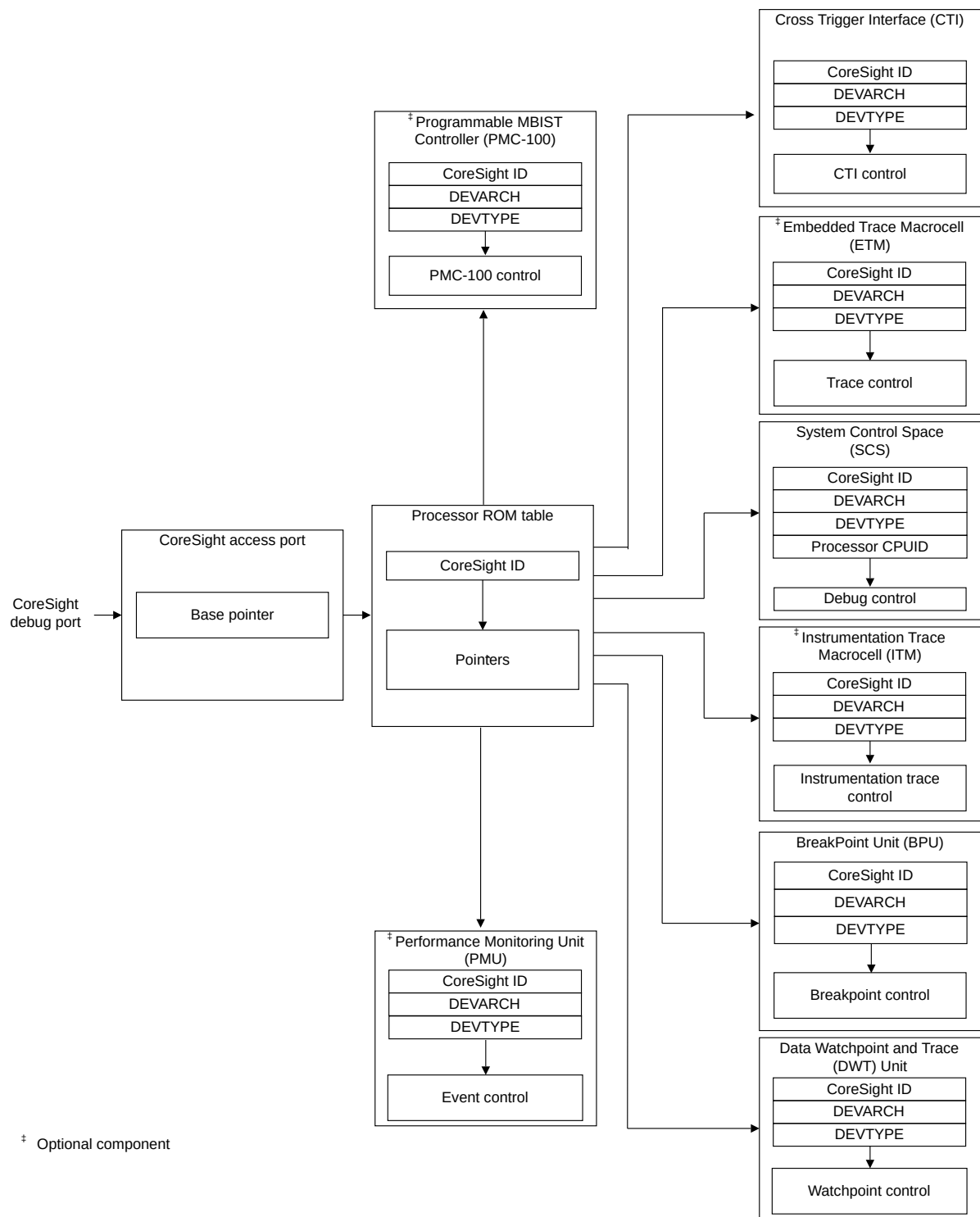
Arm® recommends that a debugger identifies and connects to the debug components using the CoreSight™ debug infrastructure.

See the *Arm® CoreSight™ System-on-Chip SoC-600 Technical Reference Manual* for more information.

Arm® recommends that a debugger follows the flow in the following figure to discover the components present in the CoreSight™ debug infrastructure. In this case, for each CoreSight™ component in the CoreSight™ system, a debugger reads:

- The peripheral and component ID registers.
- The DEVARCH and DEVTYPE registers.

**Figure 16-1: CoreSight discovery**



To identify the Cortex®-M52 processor and debug components within the CoreSight™ system, Arm® recommends that a debugger performs the following actions:

1. Locate and identify the Cortex®-M52 processor ROM table using its CoreSight™ identification.
2. Follow the pointers in the Cortex®-M52 processor ROM table to identify the presence of the following components:
  - a. *Cross Trigger Interface* (CTI)
  - b. *Embedded Trace Macrocell* (ETM)
  - c. *System Control Space* (SCS)
  - d. *Instrumentation Trace Macrocell* (ITM)
  - e. *BreakPoint Unit* (BPU)
  - f. *Data Watchpoint and Trace* (DWT) unit
  - g. *Performance Monitoring Unit* (PMU)
  - h. *Programmable MBIST Controller* (PMC-100)

### 16.1.2 Debugger actions for identifying the processor

When a debugger identifies the *System Control Space* (SCS) from its Cortex®-M52 identification, it can identify the processor and its revision number from the CPUID register in the SCS at address 0xE000ED00.

A debugger cannot rely on the Cortex®-M52 processor ROM table being the first ROM table encountered. One or more system ROM tables might be included between the access port and the processor ROM table if other CoreSight™ components are in the system. If a system ROM table is present, it can include a unique identifier for the implementation.

### 16.1.3 Processor ROM table identification and entries

The ROM table identification registers and its values that the following table shows allow debuggers to identify the processor and its debug capabilities.

The following table shows the CoreSight™ components that the Cortex®-M52 processor ROM table points to.

**Table 16-1: Cortex®-M52 processor ROM table components**

Address	Component	Reset value	Description
0xE00FF000	<i>System Control Space</i> (SCS)	0xFFFF0F03	See <a href="#">Debug identification block register summary</a>
0xE00FF004	<i>Data Watchpoint and Trace</i> (DWT)	If DWT is configured, 0xFFFF0203.  If DWT is not implemented, 0xFFFF0202.	See <a href="#">Data Watchpoint and Trace unit</a>

Address	Component	Reset value	Description
0xE00FF008	BreakPoint Unit (BPU)	If BPU is implemented, 0xFFFF03003.  If BPU is not implemented, 0xFFFF03002.	See <a href="#">BreakPoint Unit</a>
0xE00FF00C	Instrumentation Trace Macrocell (ITM)	If ITM is implemented, 0xFFFF01003.  If ITM is not implemented, 0xFFFF01002.	See <a href="#">Instrumentation Trace Macrocell</a>
0xE00FF010	Trace Port Interface Unit (TPIU)	0xFFFF41002	The TPIU is not configured inside the processor. It can be configured in the MCU layer and included in the MCU ROM table See <a href="#">Trace Port Interface Unit</a>
0xE00FF014	Embedded Trace Macrocell (ETM)	If ETM is implemented, 0xFFFF42003.  If ETM is not implemented, 0xFFFF42002.	See the <i>Arm China CoreSight™ ETM-M52 Technical Reference Manual</i>
0xE00FF018	Performance Monitoring Unit (PMU)	If PMU is implemented, 0xFFFF04003.  If PMU is not implemented, 0xFFFF04002.	See <a href="#">Performance Monitoring Unit Extension</a>
0xE00FF01C	Cross Trigger Interface (CTI)	If CTI is implemented, 0xFFFF43003.  If CTI is not implemented, 0xFFFF43002.	See <a href="#">Cross Trigger Interface</a>
0xE00FF020	Programmable MBIST Controller (PMC-100)	If PMC-100 is implemented, 0xFFFF47003.  If PMC-100 is not implemented, 0xFFFF47002.	<i>Arm® PMC-100 Technical Reference Manual</i>
0xE00FF024 - 0xE00FFFC8	Reserved	0x00000000	-
0xE00FFFC	SYSTEM ACCESS	0x00000001	See the <i>Arm® CoreSight™ Architecture Specification v3.0</i>
0xE00FFFD0 - 0xE00FFFE	Peripheral ID registers	<a href="#">Cortex-M52 processor ROM table identification values.</a>	
0xE00FFFF0 - 0xE00FFFC	Component ID registers		

The Cortex®-M52 processor ROM table entries point to the debug components of the processor. The offset for each entry is the offset of that component from the ROM table base address, 0xE00FF000.

See the *Arm® CoreSight™ Architecture Specification v3.0* for more information about the ROM table ID and component registers, and access types.

**Table 16-2: Cortex®-M52 processor ROM table identification values**

Address	Name	Type	Reset value	Description
0xE00FFFD0	PIDR4	RO	0x0000000A	See <i>Arm®v8-M Architecture Reference Manual</i> for more information.
0xE00FFFD4	PIDR5	RO	0x00000000	
0xE00FFFD8	PIDR6	RO	0x00000000	
0xE00FFFD4	PIDR7	RO	0x00000000	
0xE00FFFE0	PIDR0	RO	0x000000D2	
0xE00FFFE4	PIDR1	RO	0x00000054	
0xE00FFFE8	PIDR2	RO	0x0000000F	
0xE00FFFE4	PIDR3	RO	0x00000000	
0xE00FFF00	CIDR0	RO	0x0000000D	
0xE00FFF04	CIDR1	RO	0x00000010	
0xE00FFF08	CIDR2	RO	0x00000005	
0xE00FFF0C	CIDR3	RO	0x000000B1	

These values for the Peripheral ID registers identify this as the Cortex®-M52 processor ROM table. The Component ID registers identify this as a CoreSight™ ROM table.



The Cortex®-M52 processor ROM table only supports word-size transactions.

### 16.1.4 Debug identification block register summary

The *System Control Space* (SCS) provides a set of debug identification registers which can be used for debug-related peripheral and component identification.

The following table shows the debug identification registers and values for debugger detection. For more information, see the *Arm®v8-M Architecture Reference Manual*.

**Table 16-3: Debug identification values**

Address offset	Name	Type	Reset value	Description
0xE000EFD0	DPIDR4	RO	0x0000000A	SCS Peripheral Identification Register 4

Address offset	Name	Type	Reset value	Description
0xE000EFD4	DPIDR5	RO	0x00000000	SCS Peripheral Identification Register 5
0xE000EFD8	DPIDR6	RO	0x00000000	SCS Peripheral Identification Register 6
0xE000EFD8	DPIDR7	RO	0x00000000	SCS Peripheral Identification Register 7
0xE000EFE0	DPIDR0	RO	0x00000024	SCS Peripheral Identification Register 0
0xE000EFE4	DPIDR1	RO	0x0000005D	SCS Peripheral Identification Register 1
0xE000EFE8	DPIDR2	RO	0x0000000F	SCS Peripheral Identification Register 2
0xE000EFEC	DPIDR3	RO	0x00000000 <b>Note:</b> Bits [7:4] and [3:0] are REVAND and CMOD respectively.  The REVAND field indicates minor errata fixes specific to this design, for example metal fixes after implementation.  If the component is reusable IP, the CMOD field indicates whether you have modified the behavior of the component.  These values depend on the exact revision of the silicon as documented in <i>Arm® CoreSight™ Architecture Specification v3.0</i> .	SCS Peripheral Identification Register 3
0xE000EFF0	DCIDR0	RO	0x0000000D	SCS Component Identification Register 0
0xE000EFF4	DCIDR1	RO	0x00000090	SCS Component Identification Register 1
0xE000EFF8	DCIDR2	RO	0x00000005	SCS Component Identification Register 2
0xE000EFFC	DCIDR3	RO	0x000000B1	SCS Component Identification Register 3
0xE000EFBC	DDEVARCH	RO	0x47702A04	SCS Device Architecture Register
0xE000EFCC	DDEVTYPE	RO	0x00000000	SCS Device Type Register

### 16.1.5 Debug register summary

The following table shows the debug registers, with address, name, type, reset value, and description information for each register.

Each register is 32-bits wide. These registers are not banked between Security states or are banked between Security states on a bit by bit basis. For more information on these registers, see the *Arm®v8-M Architecture Reference Manual*.



**Table 16-4: Debug register summary**

Address	Name	Type	Reset value	Description
0xE000ED30	DFSR	RW	0x00000000 Cold reset only.	Debug Fault Status Register
0xE000EDF0	DHCSR	RW	0x00000000	Debug Halting Control and Status Register
0xE000EDF4	DCRSR	WO	0xFFFF00XX, bits [15:7] are <b>RES0</b>	Debug Core Register Selector Register
0xE000EDF8	DCRDR	RW	<b>UNKNOWN</b>	Debug Core Register Data Register
0xE000EDFC	DEMCR	RW	0x00000000	Debug Exception and Monitor Control Register
0xE000EE04	DAUTHCTRL	RW	0x00000000	Debug Authentication Control Register
0xE000EE08	DSCSR	RW	0x00030000	Debug Security Control and Status Register
0xE000EFB8	DAUTHSTATUS	RO	0x00XX00XX	Debug Authentication Status Register

## 16.2 D-AHB interface

The 32-bit *Debug AHB* (D-AHB) interface implements the AMBA® 5 AHB protocol. It can be used with a CoreSight™ AHB-AP to provide debugger access to all processor control and debug resources, and a view of memory that is consistent with that observed by load and store operations.

Accesses on the D-AHB interface are always little-endian.

Debugger accesses are distributed to the appropriate internal and external resource according to the address of the request. Accesses on the D-AHB are reflected on the TCM, *Main Interface*, *Peripheral AHB* (P-AHB), and *External Private Peripheral Bus* (EPPB) as appropriate.

### 16.2.1 Debug memory access

The Cortex®-M52 processor implements external debug interaction through a 32-bit AMBA® 5 AHB debug interface.

This interface can be integrated with a suitable CoreSight™ AHB-AP interface and provides debugger access to:

- All processor control and debug resources.
- A view of memory, which is consistent with the view that software load and store operations observe.

Accesses on the D-AHB interface always ignore the endianness attribute and do not pass through the data swizzling logic in the processor used for load and store requests. Therefore, accesses to addresses outside the PPB region observe data in the downstream memory endian format and accesses in the PPB region observe data in little-endian format.

- *Debug AHB* (D-AHB) accesses undergo security attribution and security access checks. The debug Security state depends on DHCSR.S\_SDE and the D-AHB input signal, HNONSECD,

which indicates the security level that a debug access requests. If this signal is asserted, this indicates that the transfer is Non-secure.

- D-AHB accesses are not checked against the *Memory Protection Unit* (MPU) for memory attribute checks unless the Unprivileged Debug is enabled for a debug Security state.
  - Unprivileged Debug is enabled for the secure debug state when DHCSR.S\_SUIDE is set.
  - Unprivileged Debug is enabled for the Non-secure debug state when DHCSR.S\_NSUIDE is set.
- If unprivileged debug is enabled, then the access is always treated as unprivileged, regardless of the value of the D-AHB signal bit HPROTD[1] and reported on the D-AHB interface.
  - If the debug Security state is Secure, then the D-AHB access is subject to permission checks based on regions that are defined in the Secure MPU.
  - If the debug Security state is Non-secure, then the D-AHB access is subject to permission checks based on regions that are defined in the Non-secure MPU.
- D-AHB accesses to the EPPB memory region (0xE0040000-0xE00FFFFF) must be marked as privileged, HPROT[1] HIGH, unless unprivileged invasive *Debug Access Port* (DAP) access is enabled by setting DAUTHCTRL.UIDAPEN for the debug security state. When DAUTHCTRL.UIDAPEN is set all the peripherals in the EPPB region can be accessed by non-privileged debug accesses through D-AHB except for the:
  - PMC-100 located at 0xE0046000-0xE0046FFF
  - *External Wakeup Interrupt Controller* (EWIC) located at 0xE0047000-0xE0047FFF.
  - SBIST controller located at 0xE0048000-0xE0048FFF

These regions can only be accessed with Secure privileged requests. Any non-privileged accesses returns an error on D-AHB.

- D-AHB accesses to the internal PPB region must be marked as privileged, unless unprivileged invasive DAP access is enabled by setting DAUTHCTRL.UIDAPEN for the debug Security state.
  - When DAUTHCTRL.UIDAPEN is set, many of the registers in the internal PPB region can be accessed. The exceptions are those registers which are normally accessible by unprivileged code. For example, some of the *Instrumentation Trace Macrocell* (ITM) registers and the STIR. For more information on the ITM registers, see [ITM register summary](#). For more information on STIR, see *Arm®v8-M Architecture Reference Manual*.
  - When DAUTHCTRL.UIDAPEN is not set and the debug access is unprivileged, then almost all accesses to the PPB registers get an error response. However, the registers which are normally accessible by unprivileged code cannot be accessed. For example, some of the *Instrumentation Trace Macrocell* (ITM) registers and the STIR. For more information on the ITM registers, see [ITM register summary](#). For more information on STIR, see *Arm®v8-M Architecture Reference Manual*.
- The security of a debug transaction on one of the external interfaces is determined by all of the following:
  - The access control signals.
  - The mapping of the address in the *Security Attribution Unit* (SAU) and *Implementation Defined Attribution Unit* (IDAU).
  - The internal debug state of the processor.

- The HNONSECD signal value that is associated with the D-AHB debug request.



- For more information on the DHCSR and DAUTHCTRL registers, see the *Arm®v8-M Architecture Reference Manual*.
- For more information on all the AMBA® 5 AHB-compliant D-AHB signals mentioned in this section, see the *Arm® AMBA® 5 AHB Protocol Specification*.

## 16.2.2 Debugger access memory attributes and data cache access

The memory attributes associated with debugger accesses on *Debug AHB* (D-AHB) depend on the debug access mode.

### Unprivileged Debug is not enabled

If Unprivileged Debug is not enabled, debugger accesses are not subject to the memory attributes defined by the *Memory Protection Unit* (MPU). Instead, the memory attributes used to perform a debugger access are derived from the HPROTD signal on D-AHB. The attributes are used differently depending on the memory region that is associated with the address.

The following table shows the behavior of debug accesses and dependency on HPROTD for both internal and externally memory-mapped regions when Unprivileged Debug is not enabled.

**Table 16-5: HPROTD attributes**

Region and interface	Description
CODE and SRAM regions TCM and Main interfaces	<p><b>Accesses to ITCM and DTCM</b></p> <p>is passed through to ITCMPRIV and DTCMPRIV. HPROTD[0] is ignored. ITCMASTER and HPROTD[1] DTCMASTER signals are asserted indicating a debugger access.</p> <p><b>Accesses to Main interfaces</b></p> <p>If an access is not completed in the data cache:</p> <ul style="list-style-type: none"> <li>• HPROTD[0] is ignored. All debugger accesses are performed with ARPROT[2] and AWPROT[2] set to 0.</li> <li>• is passed through to ARPROT[0] HPROTD[6:1], AWPROT[0], ARCACHE, and AWCACHE.</li> </ul> <p>ARMASR and AWMASR are asserted indicating a debugger access.</p>

Region and interface	Description
Peripheral, external RAM/Device, Vendor_SYS regions Main interfaces and <i>Peripheral AHB</i> HPROTD[1] (P-AHB) interfaces	<p><b>Accesses to P-AHB</b></p> <p>HPROTD[0] is ignored. All debugger accesses are performed with HPROTD[6:1]HPROTP[0] set to 1.</p> <p>HPROTD[6:1] is passed to P-AHB.</p> <p>HMASTERP is asserted indicating debugger access.</p> <p><b>Accesses to Main interfaces</b></p> <p>If an access is not completed in the data cache:</p> <ul style="list-style-type: none"> <li>HPROTD[0] is ignored. All debugger accesses are performed with ARPROT[2] and AWPROT[2] set to 0.</li> <li>HPROTD[6:1] is passed through to ARPROT[0], AWPROT[0], ARCACHE, and AWCACHE.</li> </ul> <p>ARMASTER and AWMMASTER are asserted indicating a debugger access.</p> <p><b>Note:</b></p> <p>The debugger access can complete in the data cache if the software has programmed the MPU to make this region cacheable.</p>
<i>Internal Private Peripheral Bus</i> (IPPB)	<ul style="list-style-type: none"> <li>HPROTD[0] is ignored.</li> <li>HPROTD[1] is used for register-specific checks.</li> <li>HPROTD[6:2] is ignored.</li> </ul> <p>PADDR31 is asserted indicating a debugger access.</p> <p>Unprivileged D-AHB accesses to privileged registers return an ERROR response on HRESPD.</p>
<i>External Private Peripheral Bus</i> (EPPB)	<ul style="list-style-type: none"> <li>HPROT[0] is ignored.</li> <li>HPROT[1] is passed through to PPROT[0].</li> <li>PADDR31 is asserted which indicates a debugger access.</li> </ul>

All debug read and write accesses marked as Normal cacheable and Non-shareable in HPROTD and outside the address regions associated with ITCM and DTCM look up the data cache if it is configured in the processor. If the address is present in the cache, for a read the data is returned without making any request on Main interfaces and for a write the cache line is updated. If the debug memory attribute is Write-through, then the data is also be written on Main interfaces. Debugger accesses never allocate lines to the cache on a miss. Debug accesses marked as Device, Non-cacheable or Normal shareable in HPROTD do not look up the data cache.

### Unprivileged Debug is enabled

If Unprivileged Debug is enabled, the HPROTD[6:1] input signals on D-AHB are ignored and the debugger accesses are always treated as unprivileged. The *Memory Protection Unit* (MPU) determines whether an access is allowed based on an unprivileged lookup request and the memory attributes associated with the memory access.

The following table shows the behavior of debug accesses and dependency on HPROTD for both internal and externally memory-mapped regions when Unprivileged Debug is enabled.

**Table 16-6: Memory attributes**

Region and interface	Description
CODE and SRAM regions TCM and Main interfaces	<p><b>Accesses to ITCM and DTCM</b></p> <p>HPROTD[1] is passed through to ITCMPRIV and DTCMPRIV. HPROTD[0] is ignored. ITCMMASTER and DTCMMASTER signals are asserted indicating a debugger access.</p> <p><b>Accesses to Main interfaces</b></p> <p>If an access is not completed in the data cache:</p> <ul style="list-style-type: none"> <li>• HPROTD[0] is ignored. All debugger accesses are performed with ARPROT[2] and AWPROT[2] set to 0.</li> <li>• HPROTD[6:1] is passed through to ARPROT[0], AWPROT[0], ARCACHE, and AWCACHE. ARMASTER and AWMMASTER are asserted indicating a debugger access.</li> </ul>
Peripheral, external RAM/Device, Vendor_SYS regions Main interfaces and <i>Peripheral AHB</i> (P-AHB) interfaces	<p><b>Accesses to P-AHB</b></p> <p>All debugger accesses are performed with HPROTP[0] set to 1. The memory attributes are passed to P-AHB. HMASTERP is asserted indicating debugger access.</p> <p><b>Accesses to Main interfaces</b></p> <p>If an access is not completed in the data cache:</p> <ul style="list-style-type: none"> <li>• All debugger accesses are performed with ARPROT[2] and AWPROT[2] set to 0.</li> <li>• The memory attributes are passed through to ARPROT[0], AWPROT[0], ARCACHE, and AWCACHE.</li> </ul> <p>ARMMASTER and AWMMASTER are asserted indicating a debugger access.</p>
<i>Internal Private Peripheral Bus</i> (IPPB)	<p>PADDR31 is asserted indicating a debugger access.</p> <p>Unprivileged access in some registers is allowed when DAUTHCTRL.UIDAPEN is set. if Unprivileged access is not allowed, an error response is returned on HRESPD.</p>
<i>External Private Peripheral Bus</i> (EPPB)	<p>PADDR31 is asserted which indicates a debugger access.</p>

### 16.2.3 Debug access security and attributes

Debugger accesses to memory and any memory-mapped registers are subject to the same security checks as data accesses generated by software running on the processor, with the security attributes set as the following:

- Request is Secure if the DHCSR.S\_SDE register field is 1 indicating secure debug is enabled and HNONSECD is LOW.
- Otherwise, the request is Non-secure.

The state of DHCSR.S\_SDE depends on the context of the debug request. If the processor is halted when it was in Secure state, then DHCSR.S\_SDE is 1, otherwise the value of the field depends on the secure access control input signal. This implies access to the secure state and memory is only available if secure invasive debug is permitted in the system.

In most of the memory regions, debugger accesses are subject to validation and attribution. This implies that the final security state of an access on the Main interface, *Peripheral AHB* (P-AHB), and *External Private Peripheral Bus* (EPPB) interfaces are set by the *Security Attribution Unit* (SAU) in the

same way as software generated accesses. The SAU blocks memory accesses which do not have the required permissions. For example, accesses to memory regions marked as Secure in the SAU if DHCSR.S\_SDE is 0 or HNONSECD is HIGH. This results in an error response on the *Debug AHB* (D-AHB) interface, but unlike accesses originating from software, a SecureFault is not raised.

There are a number of address regions associated with the *System Control Space* (SCS) and debug peripherals where the security state of the access is determined only by the HNONSECD signal and DHCSR.S\_SDE.

If the security extensions are not included in the processor, DHCSR.S\_SDE behaves as RAZ/WI, therefore all debug accesses are considered to be Non-secure.



For more information on the DHCSR register, see the *Arm®v8-M Architecture Reference Manual*. For more information on all the AMBA® 5 AHB-compliant HNONSECD signal, see the *Arm® AMBA® 5 AHB Protocol Specification*.

## 16.2.4 Debug during reset and before code execution commences

The Cortex®-M52 processor supports access to the debug and trace resource from a debug agent connected to the *Debug AHB* (D-AHB) interface when the device is in processor reset. This can be useful for setting up the debug and trace environment before any code has executed on the processor.

The following table lists the memory regions which can be accessed during processor reset. Access control and security level are determined in the same manner as debug accesses during code execution or when halted based on the authentication signals and the default SAU/IDAU regions. Any component on the EPPB, which cannot be accessed during reset, must ensure the APB PREADY and PSLVERR signals are HIGH in response to a request from the processor.

Access to all other memory areas during processor reset is **UNPREDICTABLE**.

**Table 16-7: Debug and trace registers accessible during processor reset**

Memory address range	Group	Description
0xE000E004	System Control and ID registers	ICTR register. For more information on the ICTR register, see the <i>Arm®v8-M Architecture Reference Manual</i>
0xE000ECFC		REVIDR register. <a href="#">REVIDR, Revision ID Register</a> .
0xE000ED00		CPUID register. <a href="#">CPUID, CPUID Base Register</a> .
0xE000ED30		DFSR register. For more information on the DFSR register, see <a href="#">System control register summary</a> .
0xE000ED40		ID registers. <a href="#">Identification register summary</a> .
-		
0xE000ED7F		
0xE000ED80		CCSIDR register. <a href="#">CCSIDR, Current Cache Size ID Register</a>

Memory address range	Group	Description
0xE000EDF0 - 0xE000EEFF		Debug registers. <a href="#">Debug register summary</a> .
0xE000EF40 - 0xE000EF4B		MVFR0, MVFR1, MVFR2 registers. For more information on the MVFR0, MVFR1, MVFR2 registers, see <a href="#">System control register summary</a> .
0xE000EF4B - 0xE000EFFF		Debug Identification Block. <a href="#">Debug identification block register summary</a> .
0xE0000000 - 0xE0000FFF	Instrumentation Trace Macrocell (ITM)	<a href="#">ITM register summary</a>
0xE0001000 - 0xE0001FFF	Data Watchpoint and Trace (DWT)	<a href="#">DWT register summary</a>
0xE0002000 - 0xE0002FFF	BreakPoint Unit (BPU)	<a href="#">BPU register summary</a>
0xE0003000 - 0xE0003FFF	Performance Monitoring Unit (PMU)	<a href="#">PMU register summary</a>
0xE0041000 - 0xE0041FFF	Embedded Trace Macrocell (ETM)	For more information, see the <i>Arm China CoreSight™ ETM-M52 Technical Reference Manual</i> .
0xE0042000 - 0xE0042FFF	Cross Trigger Interface (CTI)	<a href="#">CTI register summary</a>
0xE0044000 - 0xE00FEFFF	External Private Peripheral Bus (EPPB)	Access directed to Cortex®-M52 EPPB APB interface.
0xE00FF000 - 0xE00FFFFF	Processor ROM table	-

## 16.2.5 Advanced DSP debug capabilities

The Cortex®-M52 processor supports the *Digital Signal Processing* (DSP) Debug Extension to provide additional features for analyzing signal processing and compute software using the *Data Watchpoint and Trace* (DWT) and *Performance Monitoring Unit* (PMU).

For more information on the DSP Debug Extension, see the *Arm®v8-M Architecture Reference Manual* and include the following additional functionality to the processor.

The DSP debug capabilities supported are:

### DWT value mask

Value matching using the DWT comparators, DWT\_COMPn, is extended to use a mask register DWT\_VMASKn. This allows events to be selected based on sub-word values or arbitrary bitfields. This is useful for analyzing data where only part of the data word is valid.

### Halt request on PMU overflow

The processor can be configured to enter debug Halt when a PMU counter, which is configured to generate an interrupt overflow. This can be used to set up a hardware watchpoint which is triggered after a number of events have been observed in a system.

### Extended PMU events

The DSP Debug Extension defines additional PMU events specific to M-profile debug and trace operation TRCEXTOUT, CTI\_TRIGOUT and DWT\_CMPMATCH. For more information on these events, see [PMU events](#).



# 17. Performance Monitoring Unit Extension

This chapter describes the *Performance Monitoring Unit* (PMU) Extension.

## 17.1 PMU features

The Cortex®-M52 processor *Data Watchpoint and Trace* (DWT) implements the *Performance Monitoring Unit* (PMU). This enables software to get information about events that are taking place in the processor and can be used for performance analysis and system debug.

The PMU supports eight 16-bit event counters and one 32-bit cycle counter. Each event counter can count one event from a list comprising both architectural and **IMPLEMENTATION DEFINED** events. For more information on PMU events, see [PMU events](#). The PMU also supports a chain function which allows the PMU to cascade two of the 16-bit counters into one 32-bit counter. Only odd event counters support the chain feature. PMU counters increment if the appropriate bit in PMU\_CNTENSET register is set.

The Arm®v8.1-M architecture specifies that operation of the PMU counters and DWT profiling counters is mutually exclusive. The Cortex®-M52 processor uses this requirement to share the state used for the counters.

The PMU cycle counter PMU\_CCNTR is an alias of the DWT\_CYCCNT register. All derived functions of the counter are available whenever either the DWT or the PMU enables the cycle counter. If the DWT is included in the processor, DWT\_CTRL.NOCYCCNT is RAZ.

### Generating interrupts

If a counter is configured to generate an interrupt when it overflows, DEMCR.MON\_PEND is set to 1 to make a Debug Monitor exception pended with DFSR.PMU set to 1. The associated overflow bit programmed by PMU\_OVSSET and PMU\_OVSCLR indicates which counter triggered the exception. The interrupts are enabled if their corresponding bit programmed by PMU\_INTENSET and PMU\_INTENCLR is set and DEMCR.MON\_EN is 1.

### Exporting trace

The PMU can export trace whenever the lower 8 bits of the counters overflow. The PMU issues an event counter packet with the appropriate counter flag set to 1. This occurs on counter increment only, not on software or debugger write. For each counter *n*, if the lower 8 bits of that counter overflows, the associated OVN bit of the event counter packet is set. If multiple counters overflow during the same period, multiple bits might be set.

The PMU can serve as an event source for the *Cross Trigger Interface* (CTI).

For more information on the registers mentioned in this section, see the *Arm®v8-M Architecture Reference Manual*.



The *Performance Monitoring Unit* (PMU) is included if the *Data Watchpoint and Trace* (DWT) is included in the processor. For more information on performance monitoring, see the *Arm®v8.1-M Performance Monitoring User Guide Application Note*.

## 17.2 PMU events

The following table shows the events that are generated and the numbers that the *Performance Monitoring Unit* (PMU) uses to reference the events.

**Table 17-1: PMU events**

Event number	Event mnemonic	PMU event bus bit	Event name
0x0000	SW_INCR	0	Instruction architecturally executed, condition code check pass, software increment.
0x0001	L1I_CACHE_REFILL	1	L1 instruction cache linefill.
0x0003	L1D_CACHE_REFILL	2	L1 data cache linefill.
0x0004	L1D_CACHE	3	L1 data cache access.
0x0006	LD_RETIRED	4	Instruction architecturally executed, condition code check pass, load.
0x0007	ST_RETIRED	5	Instruction architecturally executed, condition code check pass, store.
0x0008	INST_RETIRED	6	Instruction architecturally executed.
0x0009	EXC_TAKEN	7	Exception taken.
0x000A	EXC_RETURN	8	Instruction architecturally executed, condition code check pass, exception return
0x000C	PC_WRITE_RETIRED	9	Instruction architecturally executed, condition code check pass, software change of the PC.
0x000D	BR_IMMED_RETIRED	10	Instruction architecturally executed, immediate branch.
0x000E	BR_RETURN_RETIRED	11	Instruction architecturally executed, condition code check pass, procedure return.
0x000F	UNALIGNED_LDST_RETIRED	12	Instruction architecturally executed, condition code check pass, unaligned load or store.
0x0010	BR_MIS_PRED	13	Mispredicted or not predicted branch speculatively executed.
0x0011	CPU_CYCLES	14	Cycle.
0x0012	BR_PRED	15	Predictable branch speculatively executed.
0x0013	MEM_ACCESS	16	Data memory access.
0x0014	L1I_CACHE	17	L1 instruction cache access.
0x0015	L1D_CACHE_WB	18	L1 data cache write-back.

Event number	Event mnemonic	PMU event bus bit	Event name
0x0019	BUS_ACCESS	19	Any beat access to the M-AXI read interface, M-AXI write interface (or MAHB code region interface and system region interface when M-AHB is configured) and any access to P-AHB interface.
0x001A	MEMORY_ERROR	20	Local memory error.
0x001D	BUS_CYCLES	22	Bus cycle asserted always for M-AHB configuration. Count the number of cycles on which the M-AXI interface is clocked.
0x001E	CHAIN	23	For an odd-numbered counter, increments when an overflow occurs on the preceding even-numbered counter on the same PE.
0x0021	BR_RETIRED	25	Instruction architecturally executed, branch.
0x0022	BR_MIS_PRED_RETIRED	26	Instruction architecturally executed, mispredicted branch.
0x0023	STALL_FRONTEND	27	If there are no instructions available from the fetch stage of the processor pipeline, the processor considers the front-end of the processor pipeline as being stalled.
0x0024	STALL_BACKEND	28	If there is an instruction available from the fetch stage of the pipeline but it cannot be accepted by the decode stage of the processor pipeline, the processor considers the back-end of the processor pipeline as being stalled.
0x0036	LL_CACHE_RD	29	Last level Data cache read.
0x0037	LL_CACHE_MISS_RD	30	Last level Data cache read miss.
0x0039	L1D_CACHE_MISS_RD	31	Level 1 Data cache read miss.
0x003C	STALL	34	No operation sent for execution.
0x0040	L1D_CACHE_RD	38	Level 1 Data cache read.
0x0100	LE_RETIRED	39	Loop end instruction architecturally executed, entry registered in the LO_BRANCH_INFO cache.
0x0104	BF_RETIRED	41	Branch future instruction architecturally executed, condition code check pass. Register an entry in the LO_BRANCH_INFO cache.  Branch future instruction is treated as NOP.
0x0108	LE_CANCEL	43	LO_BRANCH_INFO cache containing a valid loop entry cleared while not in the last iteration of the loop.
0x0109	BF_CANCEL	44	LO_BRANCH_INFO cache containing a valid BF entry cleared and associated branch not taken.  Branch future instruction is treated as NOP.
0x0114	SE_CALL_S	45	Call to secure function, resulting in security state change.
0x0115	SE_CALL_NS	46	Call to Non-secure function, resulting in security state change.
0x0118	DWT_CMPMATCH0	47	<i>Data Watchpoint and Trace</i> (DWT) comparator 0 match
0x0119	DWT_CMPMATCH1	48	DWT comparator 1 match
0x011A	DWT_CMPMATCH2	49	DWT comparator 2 match
0x011B	DWT_CMPMATCH3	50	DWT comparator 3 match
0x011C	DWT_CMPMATCH4	141	DWT comparator 4 match

Event number	Event mnemonic	PMU event bus bit	Event name
0x011D	DWT_CMPMATCH5	142	DWT comparator 5 match
0x011E	DWT_CMPMATCH6	143	DWT comparator 6 match
0x011F	DWT_CMPMATCH7	144	DWT comparator 7 match
0x0200	MVE_INST_RETIRED	51	<i>M-profile Vector Extension (MVE)</i> instruction architecturally executed
0x0204	MVE_FP_RETIRED	53	MVE floating-point instruction architecturally executed.
0x0208	MVE_FP_HP_RETIRED	55	MVE half-precision floating-point instruction architecturally executed.
0x020C	MVE_FP_SP_RETIRED	57	MVE single-precision floating-point instruction architecturally executed.
0x0214	MVE_FP_MAC_RETIRED	59	MVE floating-point multiply or multiply accumulate instruction architecturally executed.
0x0224	MVE_INT_RETIRED	61	MVE integer instruction architecturally executed.
0x0228	MVE_INT_MAC_RETIRED	63	MVE integer multiply or multiply-accumulate instruction architecturally executed.
0x0238	MVE_LDST_RETIRED	65	MVE load or store instruction architecturally executed.
0x023C	MVE_LD_RETIRED	67	MVE load instruction architecturally executed
0x0240	MVE_ST_RETIRED	69	MVE store instruction architecturally executed
0x0244	MVE_LDST_CONTIG_RETIRED	71	MVE contiguous load or store instruction architecturally executed
0x0248	MVE_LD_CONTIG_RETIRED	73	MVE contiguous load instruction architecturally executed
0x024C	MVE_ST_CONTIG_RETIRED	75	MVE contiguous store instruction architecturally executed.
0x0250	MVE_LDST_NONCONTIG_RETIRED	77	MVE non-contiguous load or store instruction architecturally executed.
0x0254	MVE_LD_NONCONTIG_RETIRED	79	MVE non-contiguous load instruction architecturally executed.
0x0258	MVE_ST_NONCONTIG_RETIRED	81	MVE non-contiguous store instruction architecturally executed.
0x025C	MVE_LDST_MULTI_RETIRED	83	MVE memory instruction targeting multiple registers architecturally executed.
0x0260	MVE_LD_MULTI_RETIRED	85	MVE memory load instruction targeting multiple registers architecturally executed.
0x0264	MVE_ST_MULTI_RETIRED	87	MVE memory store instruction targeting multiple registers architecturally executed.
0x028C	MVE_LDST_UNALIGNED_RETIRED	89	MVE unaligned memory load or store instruction architecturally executed.
0x0290	MVE_LD_UNALIGNED_RETIRED	91	MVE unaligned load instruction architecturally executed.
0x0294	MVE_ST_UNALIGNED_RETIRED	93	MVE unaligned store instruction architecturally executed.
0x0298	MVE_LDST_UNALIGNED_NONCONTIG_RETIRED	95	MVE unaligned non-contiguous load or store instruction architecturally executed.
0x02A0	MVE_VREDUCE_RETIRED	97	MVE vector reduction instruction architecturally executed.
0x02A4	MVE_VREDUCE_FP_RETIRED	99	MVE floating-point vector reduction instruction architecturally executed.
0x02A8	MVE_VREDUCE_INT_RETIRED	101	MVE integer vector reduction instruction architecturally executed.

Event number	Event mnemonic	PMU event bus bit	Event name
0x02B8	MVE_PRED	102	Cycles where one or more predicated beats architecturally executed.
0x02CC	MVE_STALL	103	Stall cycles caused by an MVE instruction.
0x02CD	MVE_STALL_RESOURCE	104	Stall cycles caused by an MVE instruction because of resource conflicts.
0x02CE	MVE_STALL_RESOURCE_MEM	105	Stall cycles caused by an MVE instruction because of memory resource conflicts.
0x02CF	MVE_STALL_RESOURCE_FP	106	Stall cycles caused by an MVE instruction because of floating-point resource conflicts.
0x02D0	MVE_STALL_RESOURCE_INT	107	Stall cycles caused by an MVE instruction because of integer resource conflicts.
0x02D3	MVE_STALL_BREAK	108	Stall cycles caused by an MVE chain break.
0x02D4	MVE_STALL_DEPENDENCY	109	Stall cycles caused by MVE register dependency.
0x4007	ITCM_ACCESS	110	<i>Instruction Tightly Coupled Memory</i> (ITCM) access from software running on the processor.
0x4008	DTCM_ACCESS	111	<i>Data Tightly Coupled Memory</i> (ITCM) access from software running on the processor.
0x4010	TRCEXTOUT0	112	<i>Embedded Trace Macrocell</i> (ETM) external output 0.
0x4011	TRCEXTOUT1	113	ETM external output 1.
0x4012	TRCEXTOUT2	114	ETM external output 2.
0x4013	TRCEXTOUT3	115	ETM external output 3.
0x4018	CTI_TRIGOUT4	116	<i>Cross Trigger Interface</i> (CTI) output trigger 4.
0x4019	CTI_TRIGOUT5	117	CTI output trigger 5.
0x401A	CTI_TRIGOUT6	118	CTI output trigger 6.
0x401B	CTI_TRIGOUT7	119	CTI output trigger 7.
0xC000	ECC_ERR	120	One or more <i>Error Correcting Code</i> (ECC) errors detected.
0xC001	ECC_ERR_MBIT	121	One or more multi-bit ECC errors detected.
0xC010	ECC_ERR_DCACHE	122	One or more ECC errors in the data cache.
0xC011	ECC_ERR_ICACHE	123	One or more ECC errors detected in the instruction cache.
0xC012	ECC_ERR_MBIT_DCACHE	124	One or more multi-bit ECC errors detected in the data cache.
0xC013	ECC_ERR_MBIT_ICACHE	125	One or more multi-bit ECC errors detected in the instruction cache.
0xC020	ECC_ERR_DTCM	126	One or more ECC errors detected in the DTCM.
0xC021	ECC_ERR_ITCM	127	One or more ECC errors detected in the ITCM.
0xC022	ECC_ERR_MBIT_DTCM	128	One or more multi-bit ECC errors detected in the DTCM.
0xC023	ECC_ERR_MBIT_ITCM	129	One or more multi-bit ECC errors detected in the ITCM.
0xC200	NWAMODE_ENTER	133	No-write allocate mode entry.
0xC201	NWAMODE	134	Write-allocate store is not allocated into the data cache due to no-write-allocate mode.
0xC300	SAHB_ACCESS	135	Read or write access on the TCM-AHB interface to the TCM.
0xC301	PAHB_ACCESS	136	Read or write access to the P-AHB write interface.

Event number	Event mnemonic	PMU event bus bit	Event name
0xC302	AXI_WRITE_ACCESS or SYS_AHB_WRITE_ACCESS	137	M-AXI configuration: Any beat access to the M-AXI write interface.  M-AHB configuration: Any write beat access to the SYS-AHB interface.
0xC303	AXI_READ_ACCESS or SYS_AHB_READ_ACCESS	138	M-AXI configuration: Any beat access to the M-AXI read interface.  M-AHB configuration: Any read beat access to the SYS-AHB interface.
0xC400	DOSTIMEOUT_DOUBLE	139	Denial of Service timeout has fired twice and caused buffers to drain to allow forward progress.
0xC401	DOSTIMEOUT_TRIPLE	140	Denial of Service timeout has fired three times and blocked the LSU to force forward progress.
0xC402	CDE_INST_RETIRED	145	CDE instruction architecturally executed.
0xC404	CDE_CX1_INST_RETIRED	147	CDE CX1 instruction architecturally executed.
0xC406	CDE_CX2_INST_RETIRED	149	CDE CX2 instruction architecturally executed.
0xC408	CDE_CX3_INST_RETIRED	151	CDE CX3 instruction architecturally executed.
0xC40A	CDE_VCX1_INST_RETIRED	153	CDE VCX1 instruction architecturally executed.
0xC40C	CDE_VCX2_INST_RETIRED	155	CDE VCX2 instruction architecturally executed.
0xC40E	CDE_VCX3_INST_RETIRED	157	CDE VCX3 instruction architecturally executed.
0xC410	CDE_VCX1_VEC_INST_RETIRED	159	CDE VCX1 Vector instruction architecturally executed.
0xC412	CDE_VCX2_VEC_INST_RETIRED	161	CDE VCX2 Vector instruction architecturally executed.
0xC414	CDE_VCX3_VEC_INST_RETIRED	163	CDE VCX3 Vector instruction architecturally executed.
0xC416	CDE_PRED	165	Cycles where one or more predicated beats of a CDE instruction architecturally executed.
0xC417	CDE_STALL	166	Stall cycles caused by a CDE instruction.
0xC418	CDE_STALL_RESOURCE	167	Stall cycles caused by a CDE instruction because of resource conflicts This event is equivalent to MVE_STALL_RESOURCE but for CDE instructions.
0xC419	CDE_STALL_DEPENDENCY	168	Stall cycles caused by a CDE register dependency. This event is equivalent to MVE_STALL_DEPENDENCY but for CDE instructions.
0xC41A	CDE_STALL_CUSTOM	169	Stall cycles caused by a CDE instruction are generated by the custom hardware.
0xC41B	CDE_STALL_OTHER	170	Stall cycles caused by a CDE instruction are not covered by the other counters.
0xC420	COD_AHB_WRITE_ACCESS	175	M-AXI configuration: Reserved as zero.  M-AHB configuration: A Write beat transfer on Code-AHB.
0xC421	COD_AHB_READ_ACCESS	176	M-AXI configuration: Reserved as zero.  M-AHB configuration: A Read beat transfer on Code-AHB.
-	Reserved	180-223	Reserved for Software Test Library. Not included in PMU.



- PMU event numbers 0-120 are architectural, and 121-140 are Cortex®-M52-specific.
- All events are exported to the external output signal EVENTBUS as a single cycle pulse, which allows system level analysis of processor performance. In normal operation the EVENTBUS is only active when DWT, ITM, PMU or ETM trace is enabled. The EVENTBUS can be activated permanently by setting ACTLR.EVENTBUSEN.

## 17.3 PMU register summary

The following table shows the *Performance Monitoring Unit* (PMU) registers. Each of these registers are 32 bits wide.

For more information on these registers, see the *Arm®v8-M Architecture Reference Manual*.

**Table 17-2: PMU register summary**

Address	Name	Type	Reset value	Description
0xE0003000-0xE000301C	PMU_EVCNTR0-7	RW	0x0000XXXX	Performance Monitoring Unit Event Counter Register
0xE000307C	PMU_CCNTR	RW	UNKNOWN	Performance Monitoring Unit Cycle Counter Register
0xE0003400-0xE000341C	PMU_EVTYPERO-7	RW	0x0000XXXX	Performance Monitoring Unit Event Type and Filter Register
0xE000347C	PMU_CCFILTR	-	-	Reserved, <b>RES0</b> .
0xE0003C00	PMU_CNTENSET	RW	0x00000000	Performance Monitoring Unit Count Enable Set Register
0xE0003C20	PMU_CNTENCLR	RW	0x00000000	Performance Monitoring Unit Count Enable Clear Register
0xE0003C40	PMU_INTENSET	RW	0x00000000	Performance Monitoring Unit Interrupt Enable Set Register
0xE0003C60	PMU_INTENCLR	RW	0x00000000	Performance Monitoring Unit Interrupt Enable Clear Register
0xE0003C80	PMU_OVSCLR	RW	0x00000000	Performance Monitoring Unit Overflow Flag Status Clear Register
0xE0003CA0	PMU_SWINC	WO	0x00000000	Performance Monitoring Unit Software Increment Register
0xE0003CC0	PMU_OVSSET	RW	0x00000000	Performance Monitoring Unit Overflow Flag Status Set Register
0xE0003E00	PMU_TYPE	RO	0x00A05F08	Performance Monitoring Unit Type Register
0xE0003E04	PMU_CTRL	RW	0x000000XX	Performance Monitoring Unit Control Register
0xE0003FB8	PMU_AUTHSTATUS	RO	0x00XX00XX	Performance Monitoring Unit Authentication Status Register
0xE0003FBC	PMU_DEVARCH	RO	0x47700A06	Performance Monitoring Unit Device Architecture Register
0xE0003FCC	PMU_DEVTYPE	RO	0x00000016	Performance Monitoring Unit Device Type Register
0xE0003FD0	PMU_PIDR4	RO	0x0000000A	Performance Monitoring Unit Peripheral Identification Register 4
0xE0003FE0	PMU_PIDR0	RO	0x00000024	Performance Monitoring Unit Peripheral Identification Register 0
0xE0003FE4	PMU_PIDR1	RO	0x0000005D	Performance Monitoring Unit Peripheral Identification Register 1
0xE0003FE8	PMU_PIDR2	RO	0x0000000F	Performance Monitoring Unit Peripheral Identification Register 2
0xE0003FEC	PMU_PIDR3	RO	0x00000000	Performance Monitoring Unit Peripheral Identification Register 3
0xE0003FF0	PMU_CIDR0	RO	0x0000000D	Performance Monitoring Unit Component Identification Register 0

Address	Name	Type	Reset value	Description
0xE0003FF4	PMU_CIDR1	RO	0x00000090	Performance Monitoring Unit Component Identification Register 1
0xE0003FF8	PMU_CIDR2	RO	0x00000005	Performance Monitoring Unit Component Identification Register 2
0xE0003FFC	PMU_CIDR3	RO	0x000000B1	Performance Monitoring Unit Component Identification Register 3



# 18. Instrumentation Trace Macrocell

This chapter describes the *Instrumentation Trace Macrocell* (ITM).

## 18.1 ITM features

The Cortex®-M52 processor optionally implements the *Instrumentation Trace Macrocell* (ITM) which has the following features.

- Trace data generation. This includes:
  - `printf` style debugging using the stimulus port registers which generate instrumentation packets.
  - Global and local timestamp packet generation.
  - Synchronization packet generation.
- Arbitration between trace packets, that is, prioritizing multiple sources and selecting a single source at a time.
  - External *Data Watchpoint and Trace* (DWT) packets and internally generated packets.
  - This arbitration is done using a fixed priority scheme of the order:
    1. Synchronization requests.
    2. Stimulus.
    3. DWT.
    4. Local and global timestamps.
- Buffering packets in the FIFO before sending them to a trace sink over an AMBA® ATB interface, which is typically a CoreSight™ *Trace Port Interface Unit* (TPIU).
- Trace flush requests from the ATB interface.

The ITM functionality is predominantly architecturally defined. However, there are some **IMPLEMENTATION SPECIFIC** features.

For information on the architecturally-defined ITM functionality, see the *Arm®v8-M Architecture Reference Manual*.

The **IMPLEMENTATION SPECIFIC** information for the Cortex®-M52 ITM is detailed in this section.

### Stimulus Ports

The ITM has 32 stimulus ports, the ITM\_STIMn registers. This implies one ITM\_TER register is included and ITM\_TPR[31:4] is RAZ/WI. For more information on these registers, see the *Arm®v8-M Architecture Reference Manual*.

The Security Extension does not require that any configuration registers are banked. The only requirement is that the trace is filtered appropriately. Therefore, the following apply.

- Both Security states share the same stimulus and configuration registers.
- No trace messages are generated when non-invasive debug is disabled.
- Secure trace messages are only generated when secure non-invasive debug is enabled.

### DWT packets

The ITM arbitrates the various packets that are generated before inserting them into the FIFO. The only exception to this are the global timestamps. *Data Watchpoint and Trace* (DWT) packets are taken one at a time in the order that DWT arbitration determines. A bus similar to an ATB bus is used between the DWT and ITM.

The DWT and ITM can generate ITM synchronization packets, global timestamps, and DSYNC pulses for synchronizing the trace stream. These are generated when ITM\_TCR.SYNCENA is first enabled and then periodically generated using the DWT synchronization packet timer. For more information on the ITM\_TCR registers, see the *Arm®v8-M Architecture Reference Manual*. The DSYNC pulse causes frame synchronization within the Cortex®-M52 *Trace Port Interface Unit* (TPIU) when connected to the DSYNC input on the unit. For more information on TPIU frame synchronization, see the *Arm® CoreSight™ Architecture Specification v3.0*.

It is also possible for a downstream CoreSight™ trace component to control when synchronization packets are generated by the ITM on ATB using the input SYNCREQI signal.

### Local timestamp, LTS

The local timestamp counter is used to create a time delta between each LTS message.

### Global timestamp, GTS

64-bit global timestamp packets can be generated from an external timer source.

### Busy flag conditions

The ITM\_TCR register includes BUSY status bit that indicates when the ITM is processing events, including all internally generated and DWT packets.

For more information on the ITM\_TCR register, see *Arm®v8-M Architecture Reference Manual*.

### Stimulus disabled bit

On read transactions, the ITM\_STIMn.FIFOREADY indicates whether the local stimulus FIFO or buffer is ready to accept data. For more information on the ITM\_STIMn register, see the *Arm®v8-M Architecture Reference Manual*.

### Processor stalling for guaranteed trace

In some cases, the processor might need to be stalled to ensure that no trace data is lost because of FIFO overflow. This optional architectural feature can be enabled or disabled using the ITCM\_TCR.STALLENA field. Using this feature might affect processor performance.

## 18.2 ITM register summary

The following table shows the *Instrumentation Trace Macrocell* (ITM) registers whose implementation is specific to this processor.

Other registers are described in the *Arm®v8-M Architecture Reference Manual*.

Depending on the implementation of your processor, the ITM registers might not be present. Any register that is configured as not present reads as zero.



- You must enable DEMCR.TRCENA before you program or use the ITM.
- If the ITM stream requires synchronization packets, you must configure the synchronization packet rate in the DWT.

**Table 18-1: ITM register summary**

Address	Name	Type	Reset	Description
0xE0000000- 0xE000007C	ITM_STIM0- ITM_STIM31	RW	0x00000002	ITM Stimulus Port Registers 0-31
0xE0000E00	ITM_TER	RW	0x00000000	ITM Trace Enable Register
0xE0000E40	ITM_TPR	RW	0x00000000	ITM_TPR, ITM Trace Privilege Register
0xE0000E80	ITM_TCR	RW	0x00000000	ITM Trace Control Register
0xE0000EF0	INT_ITREAD	RO	0x00000000	ITM_ITREAD, Integration Read Register
0xE0000EF8	INT_ITWRITE	WO	0x00000000	ITM_ITWRITE, Integration Write Register
0xE0000F00	ITM_ITCTRL	WO	0x00000000	ITM_ITCTRL, ITM Integration Mode Control Register
0xE0000FBC	ITM_DEVARCH	RO	0x47701A01	ITM CoreSight™ Device Architecture Register
0xE0000FCC	ITM_DEVTYPE	RW	0x00000043	ITM CoreSight™ Device Type Register
0xE0000FD0	ITM_PIDR4	RO	0x0000000A	ITM Peripheral identification registers
0xE0000FD4	ITM_PIDR5	RO	0x00000000	
0xE0000FD8	ITM_PIDR6	RO	0x00000000	
0xE0000FDC	ITM_PIDR7	RO	0x00000000	
0xE0000FE0	ITM_PIDR0	RO	0x00000024	
0xE0000FE4	ITM_PIDR1	RO	0x0000005D	
0xE0000FE8	ITM_PIDR2	RO	0x0000000F	
0xE0000FEC	ITM_PIDR3	RO	0x00000000	
0xE0000FF0	ITM_CIDR0	RO	0x0000000D	ITM Component identification registers
0xE0000FF4	ITM_CIDR1	RO	0x00000090	
0xE0000FF8	ITM_CIDR2	RO	0x00000005	
0xE0000FFC	ITM_CIDR3	RO	0x000000B1	

ITM registers are fully accessible in privileged mode.



In user mode:

- All registers can be read.
- Only the Stimulus registers and Trace Enable registers can be written, and only when the corresponding Trace Privilege Register bit is set.
- Writes to registers other than the Stimulus registers and Trace Enable registers are invalid and they are ignored.

When the Security Extension is included in the Cortex®-M52 processor and if Secure non-invasive debug authentication is not enabled, writes to the Stimulus registers from the software running in Secure state are ignored.

## 18.3 ITM\_TPR, ITM Trace Privilege Register

The ITM\_TPR enables an operating system to control the stimulus ports that are accessible by user code.

### Usage constraints

You can only write to this register in privileged mode.

### Configurations

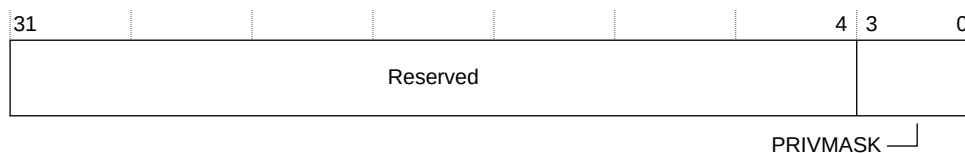
This register is available if the ITM is configured in your implementation.

### Attributes

See [ITM register summary](#) for more information.

The following figure shows the ITM\_TPR bit assignments.

**Figure 18-1: ITM\_TPR bit assignments**



The following table shows the ITM\_TPR bit assignments.

**Table 18-2: ITM\_TPR bit assignments**

Bits	Name	Function
[31:4]	-	Reserved, <b>RES0</b> .
[3:0]	PRIVMASK	Bit mask to enable tracing on ITM stimulus ports:  <div> <div>Bit[0]</div> <div>Bit[1]</div> <div>Bit[2]</div> <div>Bit[3]</div> </div> <div> <div>Stimulus ports [7:0].</div> <div>Stimulus ports [15:8].</div> <div>Stimulus ports [23:16].</div> <div>Stimulus ports [31:24].</div> </div>

## 18.4 ITM\_ITCTRL, ITM Integration Mode Control Register

The ITM\_ITCTRL controls whether the trace unit is in integration mode.

### Usage

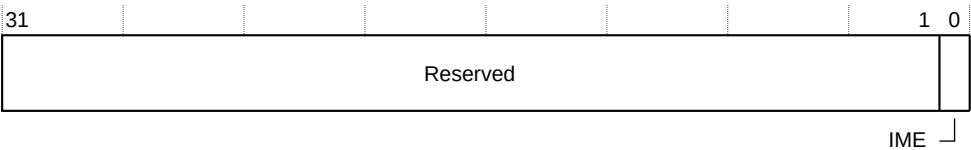
#### constraints

- Accessible from the memory-mapped interface or from an external agent such as a debugger.

- 
- Arm® recommends that you perform a debug reset after using integration mode. This register is write only and is only accessible in privilege mode.
- Configurations** Available in all configurations.
- Attributes** See [ITM register summary](#) for more information.

The following figure shows the ITM\_ITCTRL bit assignments.

Figure 18-2: ITM\_ITCTRL bit assignments



The following table shows the ITM\_ITCTRL bit assignments.

Table 18-3: ITM\_ITCTRL bit assignments

Bits	Name	Function
[31:1]	-	Reserved, <b>RES0</b> .
[0]	IME	Integration mode enable bit. The possible values are:  <b>0</b> The trace unit is not in integration mode. <b>1</b> The trace unit is in integration mode. This mode enables: <ul style="list-style-type: none"><li>• A debug agent to perform topology detection.</li><li>• SoC test software to perform integration testing.</li></ul>

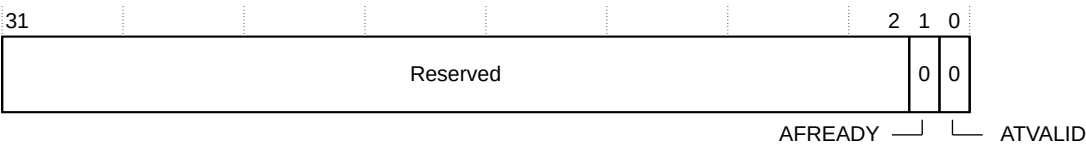
## 18.5 ITM\_ITWRITE, Integration Write Register

ITM\_ITWRITE is used for integration testing.

- Usage constraints** This register is write only, and all reads are ignored. When ITM\_ITCTRL.IME is not set and the processor is in privilege mode, then you can still write to this register. However, if the processor is not in privilege mode, then you cannot write to this register.
- Configurations** This register is:
  - Only present in integration mode, when ITM\_ITCTRL.IME is set to 1.
  - Available in all configurations.
- Attributes** See [ITM register summary](#) for more information.

The following figure ITM\_ITWRITE shows the bit assignments.

Figure 18-3: ITM\_ITWRITE bit assignments



The following table shows the ITM\_ITWRITE bit assignments.

Table 18-4: ITM\_ITWRITE bit assignments

Bits	Name	Function
[31:2]	Reserved	<b>RES0</b>
[1]	AFREADY	When ITM_ITCTRL.IME is set, the value of this bit determines the value of AFREADYI. For more information on AFREADYI, see <a href="#">ITM interface signals</a> .
[0]	ATVALID	When ITM_ITCTRL.IME is set, when this bit is read, it returns the value of ATVALIDI. For more information on ATFVALIDI, see <a href="#">ITM interface signals</a> .

## 18.6 ITM\_ITREAD, Integration Read Register

ITM\_ITREAD is used for integration test.

- Usage constraints

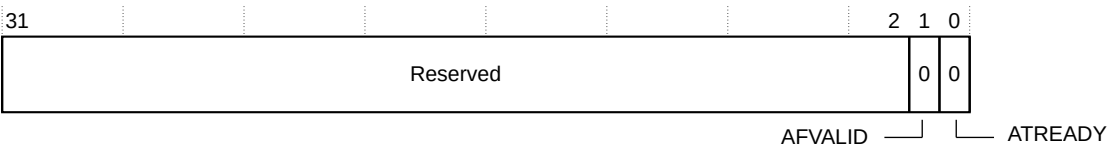
This is a read-only register, and all writes are ignored. If ITM\_ITCTRL.IME has not been set at all, then ITM\_ITREAD.AFVALID and ITM\_ITREAD\_ATREADY bits return zero. However, in the case where ITM\_ITCTRL.IME has been set at least once before, but is currently not set, then ITM\_ITREAD.AFVALID and ITM\_ITREAD.ATREADY return the previously stored AFVALIDI and ATREADYI values respectively.
- Configurations

This register is:
  - Only present in integration mode, when ITM\_ITCTRL.IME is set to 1.
  - Available in all configurations.
- Attributes

See [ITM register summary](#) for more information.

The following figure ITM\_ITREAD shows the bit assignments.

Figure 18-4: ITM\_ITREAD bit assignments



The following table shows the ITM\_ITREAD bit assignments.

**Table 18-5: ITM\_ITREAD bit assignments**

Bits	Name	Function
[31:2]	Reserved	<b>RES0</b>
[1]	AFVALID	When ITM_ITCTRL.IME is set, when this bit is read, it returns the value of AFVALIDI. When ITM_ITCTRL.IME is not set, this bit returns zero. For more information on AFVALIDI, see <a href="#">ITM interface signals</a> .
[0]	ATREADY	When ITM_ITCTRL.IME is set, when this bit is read, it returns the value of ATREADYI. When ITM_ITCTRL.IME is not set, this bit returns zero. For more information on ATREADYI, see <a href="#">ITM interface signals</a> .

# 19. Data Watchpoint and Trace unit

This chapter describes the *Data Watchpoint and Trace* (DWT) unit.

## 19.1 DWT features

The Cortex®-M52 processor *Data Watchpoint and Trace* (DWT) unit has the following features:

- Watchpoints
- Data tracing
- Trace control signaling based on comparator match which can be used to control the optional *Embedded Trace Macrocell* (ETM) and *Cross Trigger Interface* (CTI) if they are configured in the processor
- *Program Counter* (PC) tracing
- Cycle count matching
- Additional PC sampling:
  - PC sample trace output as a result of a cycle count event
  - External PC sampling using a PC sample register
- Exception tracing
- Match event tracing
- Performance profiling counters
- An implementation of the *Performance Monitoring Unit* (PMU), sharing the event counters with the regular Cortex®-M52 profiling counters. PMU events can be traced through the *Instrumentation Trace Macrocell* (ITM) and can be used to raise interrupts
- Support for the *Digital Signal Processing* (DSP) extension

The DWT receives data transactions and instruction execution information from the processor core. Exception information and core profiling information is also delivered to the DWT from the processor core. The DWT comparators can be configured for two simultaneous data value comparisons.

The DWT compares instruction and data information using the comparators that are programmed according to the debug architecture. The results of these comparisons and any profiling counter and exception information are passed to the packet generator so it can generate, buffer, and arbitrate packets to be sent to the ITM.

Additional functionality includes ETM triggers using the CMPMATCH signals and invasive watchpoint debugging.



According to the architecture, all DWT debug events are asynchronous and are not recognized on the instruction which caused the event. Therefore the DWT PC-matching functionality cannot be used to implement breakpoints in the processor.

The Cortex®-M52 processor DWT supports tracing of exceptions using an interface to the processor. The exception state information is determined from the processor core exception control signals which indicate the following events:

- Idle.
- Exception entry.
- Exception exit.
- Exception return.

When exception trace is enabled in DWT\_CTRL.EXTRCENA, these events cause the DWT to output exception packets to the ITM.

---

Data Trace Data Address packets are generated when there is a data address range match and if the comparator pair has been programmed accordingly. For more information on Data Trace Data Address packets, see the *Arm®v8-M Architecture Reference Manual*.



Note

When there is a data address range match where the address of the first access is below the lower limit of the programmed address range, the Data Trace Data Address packet that is generated contains the address of the first access instead of the address of the first matching access. In this case, however, debugger tools can reconstruct the address of the first matching access by considering the following:

- A Data Trace Data Address packet has been generated, implying that there is a data address range match.
- The data address that is stored in the Data Trace Data Address packet is lower than the programmed lower range limit.

Therefore, the debugger tool can reconstruct the address of the first matching access to be equal to the programmed lower limit value of the address range.

---

## 19.2 DWT debug access control

The *Data Watchpoint and Trace* (DWT) features are dependent on whether DEMCR.TRCENA is set to enable trace and whether invasive or non-invasive debug is allowed at a given security level.

Invasive debug could possibly change the state of the processor. Non-invasive debug guarantees not to interfere or change the state of the processor. Both invasive and non-invasive debug provide memory access control, but there are certain restrictions on memory access control for non-invasive debug. For more information, see the *Arm®v8-M Architecture Reference Manual*.

The following table lists the DWT features for the possible invasive and non-invasive debug options.

**Table 19-1: DWT debug access control**

DEMCR.TRCENA	Invasive debug	Non-invasive debug	DWT features
0	Disabled	Disabled	No DWT watchpoints.
			Debugger accesses are blocked, except for CoreSight™ ID registers.
			Profiling and <i>Performance Monitoring Unit</i> (PMU) counters disabled. The DWT_CYCCNT (cycle counter) is disabled.
			Exception trace disabled.
			All comparators are disabled. This implies that there is no data and instruction trace.
			DWT_PCSR reads 0xFFFFFFFF.
	-	Enabled	No DWT watchpoints.
			Profiling and PMU counters disabled. The DWT_CYCCNT (cycle counter) is disabled.
			Exception trace disabled.
			All comparators are disabled. This implies that there is no data and instruction trace.
1	Disabled	Disabled	No DWT watchpoints.
			Debugger accesses are blocked, except for CoreSight™ ID registers.
			Profiling and PMU counters disabled. The DWT_CYCCNT (cycle counter) is not disabled.
			Exception trace disabled.
			All comparators are disabled. This implies that there is no data and instruction trace.
			DWT_PCSR reads 0xFFFFFFFF.
	Disabled	Enabled	No DWT watchpoints.
			Profiling and PMU counters enabled.
			Exception trace enabled.
			All comparators are enabled. This implies that there is data and instruction trace.
	Enabled	Enabled	Full DWT functionality.



For a description of DEMCR and DWT\_PCSR, see the *Arm®v8-M Architecture Reference Manual*.

Note

## 19.3 DWT comparators

The *Data Watchpoint and Trace* (DWT) comparators offer various features which are adjusted based on the number of comparators supported in the Cortex®-M52 processor configuration.

The Arm® debug architecture includes the facility to match on any address range by linking two comparators together, one marking the start of the range and the other marking the end of the range.

The following table shows the two comparator configuration, also referred to as the reduced set configuration.

**Table 19-2: Two comparators configuration**

Comparator number	Instruction address matching	Data address matching	Cycle count matching	Data value matching	Supports linking?
0	Yes	Yes	Yes	No	No
1	Yes	Yes	No	Yes	Yes

The following table shows the four comparator configuration, also referred to as the mid set configuration.

**Table 19-3: Four comparators configuration**

Comparator number	Instruction address matching	Data address matching	Cycle count matching	Data value matching	Supports linking?
0	Yes	Yes	Yes	No	No
1	Yes	Yes	No	No	Yes
2	Yes	Yes	No	No	No
3	Yes	Yes	No	Yes	Yes

The following table shows the eight comparator configuration, also referred to as the full set configuration.

**Table 19-4: Eight comparators configuration**

Comparator number	Instruction address matching	Data address matching	Cycle count matching	Data value matching	Supports linking?
0	Yes	Yes	Yes	No	No
1	Yes	Yes	No	Yes	Yes
2	Yes	Yes	No	No	No
3	Yes	Yes	No	Yes	Yes
4	Yes	Yes	No	No	No
5	Yes	Yes	No	No	Yes
6	Yes	Yes	No	No	No
7	Yes	Yes	No	No	Yes



Note

- If linking is enabled on comparator 1, then there is no support for cycle count matching.
- For more information on determining the result of a comparator match that is done using the DWT\_FUNCTION registers, see the *Arm®v8-M Architecture Reference Manual*.
- If the Cortex®-M52 processor is configured to include the *Embedded Trace Macrocell*, then the DWT can control trace start and stop functionality based on the comparator results using the CMPMATCH event, which is programmed using the DWT\_FUNCTION registers.
- DBGLVL parameter determines whether two, four, or eight DWT comparators are included.

## 19.4 Cycle counter and profiling counters

The Cortex®-M52 DWT supports a cycle counter and profiling counters.

### Cycle counter

When enabled in DWT\_CTRL, the 32-bit cycle counter, DWT\_CYCCNT, increments each cycle unless the processor is in debug halt state. When the cycle counter is disabled, all functionality associated with the cycle counter is also disabled.

If the processor includes support for the Security Extension then the DWT\_CTRL.CYCDISS bit field disables the cycle counter increment when the processor is executing secure code. This can be useful for generating CPI measurements for Non-secure applications.

### Profiling counters

The profiling counters can be configured to generate events on overflow using DWT\_CTRL fields.

#### CPI Counter (DWT\_CPICNT)

The 8-bit CPI counter is incremented for every additional cycle, that is, greater than one taken to execute a non-load or store instruction. This counter must also be incremented for every cycle where fetch is stalled.

#### Exception Overhead Counter (DWT\_EXCCNT)

The 8-bit Exception Overhead Counter is incremented for every cycle associated with exception entry and return. This includes stacking, unstacking, and preemption and tail-chaining, in cases where additional registers must be stacked due to a change in Security state between exceptions. Register stacking associated with floating-point lazy context saving is also included in this counter.

#### Sleep Overhead Counter (DWT\_SLEPCNT)

The 8-bit Sleep Overhead Counter is incremented for every cycle associated for power saving. For example, WFI and WFE exceptions.

### Load-Store Counter (DWT\_LSUCNT)

The 8-bit Load-Store Counter is incremented for every additional cycle that is greater than one taken to execute a load-store instruction.

### Fold Counter (DWT\_FOLDCNT)

The 8-bit Fold Counter counts folded instructions and increments for every instruction executed in zero cycles. All folded instructions are dual-issued. For example, for a dual-issued pair of instructions, the counter increments by one to reflect this.

## 19.5 DWT register summary

The following table shows the *Data Watchpoint and Trace* (DWT) registers. Depending on the implementation of your processor, some of these registers might not be present. Any register that is configured as not present reads as zero.

**Table 19-5: DWT register summary**

Address	Name	Type	Reset value	Description
0xE0001000	DWT_CTRL	RW	Possible reset values are: <ul style="list-style-type: none"> <li>0x48000000: Reduced DWT with no <i>Instrumentation Trace Macrocell</i> (ITM) trace.</li> <li>0x40000000: Reduced DWT with ITM trace.</li> <li>0x88000000: Full DWT with no ITM trace.</li> <li>0x80000000: Full DWT with ITM trace.</li> </ul>	DWT Control Register
0xE0001004	DWT_CYCCNT	RW	UNKNOWN	DWT Cycle Count Register
0xE0001008	DWT_CPICNT	RW	0x000000XX	DWT CPI Count Register
0xE000100C	DWT_EXCCNT	RW	0x000000XX	DWT Exception Overhead Count Register
0xE0001010	DWT_SLEPCNT	RW	0x000000XX	DWT Sleep Count Register
0xE0001014	DWT_LSUCNT	RW	0x000000XX	DWT LSU Count Register

Address	Name	Type	Reset value	Description
0xE0001018	DWT_FOLD_CNT	RW	0x000000XX	DWT Folded-instruction Count Register
0xE000101C	DWT_PCSR	RO	UNKNOWN	DWT Program Counter Sample Register
0xE0001020	DWT_COMP0	RW	UNKNOWN	DWT Comparator Register 0
0xE0001028	DWT_FUNCTION0	RW	0x58000000	DWT Function Register 0
0xE0001030	DWT_COMP1	RW	UNKNOWN	DWT Comparator Register 1
0xE0001038	DWT_FUNCTION1	RW	0xF0000000	DWT Function Register 1
0xE0001040	DWT_COMP2	RW	UNKNOWN	DWT Comparator Register 2
0xE0001048	DWT_FUNCTION2	RW	0x50000000	DWT Function Register 2
0xE0001050	DWT_COMP3	RW	UNKNOWN	DWT Comparator Register 3
0xE0001058	DWT_FUNCTION3	RW	Possible reset values are: <ul style="list-style-type: none"> <li>0xD0000000: Reduced DWT</li> <li>0xF0000000: Full DWT</li> </ul>	DWT Function Register 3
0xE0001060	DWT_COMP4	RW	UNKNOWN	DWT Comparator Register 4 Can only be used for watchpoint and CMPMATCH triggers. Does not include data value or Trace support.
0xE0001068	DWT_FUNCTION4	RW	0x50000000	DWT Function Register 4
0xE0001070	DWT_COMP5	RW	UNKNOWN	DWT Comparator Register 5 Can only be used for watchpoint and CMPMATCH triggers. Does not include data value or Trace support.  Can be linked to DWT_COMP4 to perform linked comparisons when DBGLVL=2.
0xE0001078	DWT_FUNCTION5	RW	0xD0000000	DWT Function Register 6
0xE0001080	DWT_COMP6	RW	UNKNOWN	DWT Comparator Register 6 Can only be used for watchpoint and CMPMATCH triggers. Does not include data value or Trace support.
0xE0001088	DWT_FUNCTION6	RW	0x50000000	DWT Function Register 6
0xE0001090	DWT_COMP7	RW	UNKNOWN	DWT Comparator Register 7 Can only be used for watchpoint and CMPMATCH triggers. Does not include data value or Trace support.  Can be linked to DWT_COMP6 to perform linked comparisons when DBGLVL=2.
0xE0001098	DWT_FUNCTION7	RW	0xD0000000	DWT Function Register 7
0xE000103C	DWT_VMASK1	RW	UNKNOWN	DWT Comparator Value Mask Register 0-14 DWT_VMASK3 is only present when DBGLVL=2. That is, when the processor is configured to have full set debug functionality, with eight DWT and eight BPU comparators.  A maximum of two DWT_VMASK registers can be active. When DBGLVL=2, the comparators support two data value comparisons. Only comparators that can perform data value matching have corresponding DWT_VMASK registers. For more information on comparator configuration, see <a href="#">DWT comparators</a>
0xE000105C	DWT_VMASK3	RW		

Address	Name	Type	Reset value	Description
0xE0001FBC	DWT_DEVARCH	RO	0x47711A02	DWT Device Type Architecture register
0xE0001FCC	DWT_DEVTYP	RO	0x00000000	DWT Device Type Identifier register
0xE0001FD0	DWT_PIDR4	RO	0x0000000A	DWT Peripheral identification registers 0-7
0xE0001FD4	DWT_PIDR5	RO	0x00000000	
0xE0001FD8	DWT_PIDR6	RO	0x00000000	
0xE0001FDC	DWT_PIDR7	RO	0x00000000	
0xE0001FE0	DWT_PIDR0	RO	0x00000024	
0xE0001FE4	DWT_PIDR1	RO	0x0000005D	
0xE0001FE8	DWT_PIDR2	RO	0x0000000F	
0xE0001FEC	DWT_PIDR3	RO	0x00000000	
0xE0001FF0	DWT_CIDR0	RO	0x0000000D	DWT Component identification registers 0-3
0xE0001FF4	DWT_CIDR1	RO	0x00000090	
0xE0001FF8	DWT_CIDR2	RO	0x00000005	
0xE0001FFC	DWT_CIDR3	RO	0x000000B1	

DWT registers are described in the *Arm®v8-M Architecture Reference Manual*.



- DWT\_COMP4, DWT\_COMP5, DWT\_COMP6, and DWT\_COMP7 can only be used for watchpoint and `COMPATCH` and triggers and do not include data value or Trace support.
- DWT\_COMP5 can be linked to DWT\_COMP4 to perform linked comparisons when `DBG_LVL=2`.
- DWT\_COMP7 can be linked to DWT\_COMP6 to perform linked comparisons when `DBG_LVL=2`.

## 20. Cross Trigger Interface

This chapter describes the *Cross Trigger Interface* (CTI).

### 20.1 CTI features

The Cortex®-M52 processor *Cross Trigger Interface* (CTI) enables the processor debug logic and the *Embedded Trace Macrocell* (ETM) to interact with each other and with additional CoreSight™ debug and trace components in the system. This is done using trigger events across a standard interface and protocol. This allows software running on Cortex®-M52 to be debugged efficiently in both single processor systems and larger systems containing multiple processors.

The CTI is connected to a number of trigger inputs and outputs. The Cortex®-M52 CTI includes an external CTI channel interface with four input and four output channels. The input channel must be synchronous to CLKIN. The following figure shows the processor, ETM, CTI, and the available trigger input and output connections.

---

If the processor is configured with an ETM:

- Triggers 0-3 are connected to the event input and output signals.
- Up to a maximum of three *Data Watchpoint and Trace* (DWT) comparators (0, 1, and 2) can trigger events using CMPMATCH.



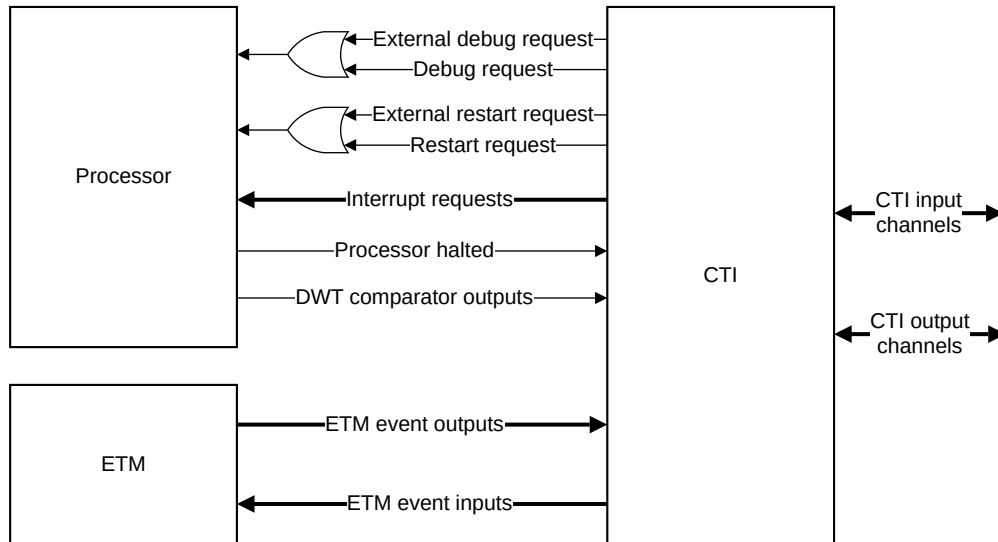
If the processor is not configured with an ETM, then the relevant triggers are not connected to the event input and output signals, and they are tied LOW.

When eight DWT comparators are configured in the processor, comparators 4, 5, 6, and 7 outputs are not used by the CTI.

---



**Figure 20-1: Cortex®-M52 processor CTI trigger connections**



The following tables show the Cortex®-M52 processor CTI trigger signals assignment.

**Table 20-1: Cortex®-M52 processor CTI input trigger signals assignment**

Signal	Description	Connection	Acknowledge, handshake
CTITRIGIN[7]	Unused	ETM to CTI <b>Note:</b> If the ETM is not included, bits [4] and [5] are unused and tied LOW.	Pulsed
CTITRIGIN[6]	Unused		
CTITRIGIN[5]	ETM Event Output 1		
CTITRIGIN[4]	ETM Event Output 0 or DWT Comparator Output 3		
CTITRIGIN[3]	DWT Comparator Output 2	Processor to CTI	
CTITRIGIN[2]	DWT Comparator Output 1		
CTITRIGIN[1]	DWT Comparator Output 0		
CTITRIGIN[0]	Processor halted		

**Table 20-2: Cortex®-M52 processor CTI output trigger signals assignment**

Signal	Description	Connection	Acknowledge, handshake
CTITRIGOUT[7]	ETM Event Input 3	CTI to ETM <b>Note:</b> If the ETM is not included, bits[7:4] are unused and the output is left untied.	Pulsed
CTITRIGOUT[6]	ETM Event Input 2		
CTITRIGOUT[5]	ETM Event Input 1		
CTITRIGOUT[4]	ETM Event Input 0		
CTITRIGOUT[3]	Interrupt Request 1	CTI to system	Acknowledged by software writing to CTIINTACK register in the interrupt service routine.
CTITRIGOUT[2]	Interrupt Request 0		
CTITRIGOUT[1]	Processor Restart Request	CTI to processor	Processor restarted

Signal	Description	Connection	Acknowledge, handshake
CTITRIGOUT[0]	Processor Debug Halt Request		Acknowledged by the debugger writing to the CTIINTACK register.



The ETM is available for download when you license Cortex®-M52 processor IP. For more information on the ETM, see *Arm China CoreSight™ ETM-M52 Technical Reference Manual*.

## 20.2 CTI register summary

The following table shows the *Cross Trigger Interface* (CTI) programmable registers, with address offset, type, and reset value for each register.

**Table 20-3: CTI register summary**

Address	Name	Type	Reset value	Description
0xE0042000	CTI_CONTROL	RW	0x00000000	CTI_CONTROL, CTI Control Register
0xE0042010	CTI_INTACK	WO	0x0000000X	CTI_INACK, CTI Interrupt Acknowledge Register
0xE0042014	CTI_APPSET	RW	0x00000000	CTI_APPSET, CTI Application Channel Set Register
0xE0042018	CTI_APPCLEAR	WO	0x00000000	CTI_APPCLR, CTI Application Channel Clear Register
0xE004201C	CTI_APPPULSE	WO	0x00000000	CTI_APPPULSE, CTI Application Channel Pulse Register
0xE0042020	CTI_INEN0	RW	0x00000000	CTI_INEN<n>, n=0-5, CTI Trigger <n> to Channel Enable Register
0xE0042024	CTI_INEN1	RW	0x00000000	
0xE0042028	CTI_INEN2	RW	0x00000000	
0xE004202C	CTI_INEN3	RW	0x00000000	
0xE0042030	CTI_INEN4	RW	0x00000000	
0xE0042034	CTI_INEN5	RW	0x00000000	
0xE0042038	CTI_INEN6	-	-	Reserved
0xE004203C	CTI_INEN7	-	-	
0xE00420A0	CTI_OUTEN0	RW	0x00000000	CTI_OUTEN<n>, n=0-7, CTI Channel <n> to Trigger Enable Register
0xE00420A4	CTI_OUTEN1	RW	0x00000000	
0xE00420A8	CTI_OUTEN2	RW	0x00000000	
0xE00420AC	CTI_OUTEN3	RW	0x00000000	
0xE00420B0	CTI_OUTEN4	RW	0x00000000	
0xE00420B4	CTI_OUTEN5	RW	0x00000000	
0xE00420B8	CTI_OUTEN6	RW	0x00000000	
0xE00420BC	CTI_OUTEN7	RW	0x00000000	
0xE0042130	CTI_TRIGINSTATUS	RO	UNKNOWN	CTI_TRIGINSTATUS, CTI Trigger Input Status Register
0xE0042134	CTI_TRIGOUTSTATUS	RO	UNKNOWN	CTI_TRIGOUTSTATUS, CTI Trigger Output Status Register
0xE0042138	CTI_CHINSTATUS	RO	0x0000000X	CTI_CHINSTATUS, CTI Channel Input Status Register

Address	Name	Type	Reset value	Description
0xE004213C	CTI_CHOUTSTATUS	RO	0x0000000X	CTI_CHOUTSTATUS, CTI Channel Output Status Register
0xE0042140	CTI_CHANNELGATE	RW	0x0000000F	CTI_CHANNELGATE, CTI Channel Gate Register
0xE0042EE4	CTI_ITCHOUT	WO	0x00000000	CTI_ITCHOUT, Integration Test Channel Output Register
0xE0042EE8	CTI_ITTRIGOUT	WO	0x00000000	CTI_ITTRIGOUT, Integration Test Trigger Output Register
0xE0042EF4	CTI_ITCHIN	RO	0x00000000	CTI_ITCHIN, Integration Test Channel Input Register
0xE0042EF8	CTI_ITTRIGIN	RO	0x00000000	CTI_ITTRIGIN, Integration Test Trigger Input Register
0xE0042F00	CTI_ITCONTROL	RW	0x00000000	CTI_ITCONTROL, Integration Mode Control Register
0xE0042FBC	CTI_DEVARCH	RO	0x47701A14	CTI_DEVARCH, Device Architecture Register
0xE0042FC8	CTI_DEVID	RO	0x01040800	CTI_DEVID, Device Configuration Register
0xE0042FCC	CTI_DEVTYPE	RO	0x00000014	CTI_DEVTYPE, Device Type Identifier Register
0xE0042FD0	CTI_PIDR4	RO	0x0000000A	CTI_PIDR4, Peripheral Identification Register 4
0xE0042FD4	CTI_PIDR5	RO	0x00000000	CTI_PIDR5, Peripheral Identification Register 5
0xE0042FD8	CTI_PIDR6	RO	0x00000000	CTI_PIDR6, Peripheral Identification Register 6
0xE0042FDC	CTI_PIDR7	RO	0x00000000	CTI_PIDR7, Peripheral Identification Register 7
0xE0042FE0	CTI_PIDR0	RO	0x00000024	CTI_PIDR0, Peripheral Identification Register 0
0xE0042FE4	CTI_PIDR1	RO	0x0000005D	CTI_PIDR1, Peripheral Identification Register 1
0xE0042FE8	CTI_PIDR2	RO	0x0000000F	CTI_PIDR2, Peripheral Identification Register 2
0xE0042FEC	CTI_PIDR3	RO	0x00000000	CTI_PIDR3, Peripheral Identification Register 3
0xE0042FF0	CTI_CIDR0	RO	0x0000000D	CTI_CIDR0, Component Identification Register 0
0xE0042FF4	CTI_CIDR1	RO	0x00000090	CTI_CIDR1, Component Identification Register 1
0xE0042FF8	CTI_CIDR2	RO	0x00000005	CTI_CIDR2, Component Identification Register 2
0xE0042FFC	CTI_CIDR3	RO	0x000000B1	CTI_CIDR3, Component Identification Register 3

## 20.3 CTI\_CONTROL, CTI Control Register

The CTI\_CONTROL register enables and disables the *Cross Trigger Interface* (CTI).

### Usage constraints

Access is only allowed from privileged code. Unprivileged access results in a BusFault being raised.

### Configurations

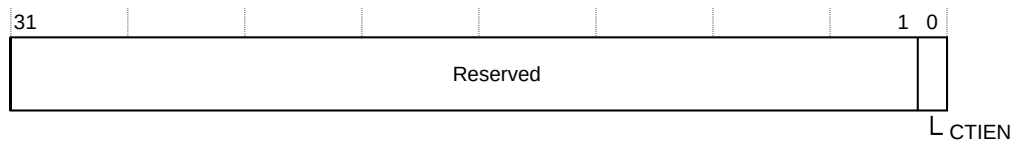
This register is always implemented when the CTI is included.

### Attributes

This is a 32-bit register. See [CTI register summary](#) for more information.

The following figure shows the CTI\_CONTROL bit assignments.

**Figure 20-2: CTI\_CONTROL bit assignments**



The following table describes the CTI\_CONTROL bit assignments.

**Table 20-4: CTI\_CONTROL bit assignments**

Field	Name	Type	Description
[31:1]	Reserved	-	<b>RES0</b>
[0]	CTIEN	RW	<p>Enable control.</p> <p><b>0</b> CTI disabled. <b>1</b> CTI enabled.</p> <p>The reset value is 0b0.</p>

## 20.4 CTI\_INACK, CTI Interrupt Acknowledge Register

The CTI\_INACK register is a software acknowledge for trigger outputs. This register is a bit map that allows selective clearing of trigger output events.

### Usage constraints

Access is only allowed from privileged code. Unprivileged access results in a BusFault being raised.

### Configurations

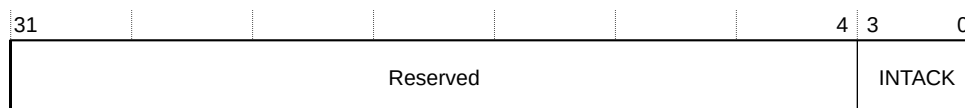
This register is always implemented when the CTI is included.

### Attributes

This is a 32-bit register. See [CTI register summary](#) for more information.

The following figure shows the CTI\_INACK bit assignments.

**Figure 20-3: CTI\_INACK bit assignments**



The following table describes the CTI\_INACK bit assignments.



## 20.6 CTI\_APPCLR, CTI Application Channel Clear Register

The CTI\_APPCLR register allows software to clear any channel output. Software can use this register to clear a channel event instead of a hardware source on a trigger input.

### Usage constraints

Access is only allowed from privileged code. Unprivileged access results in a BusFault being raised.

### Configurations

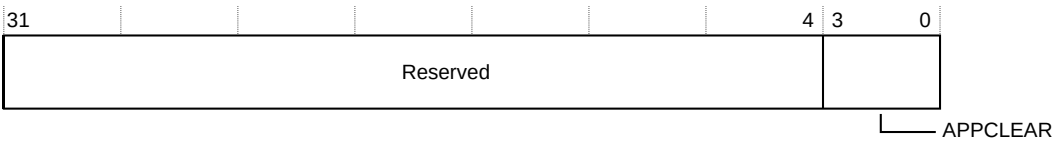
This register is always implemented when the CTI is included.

### Attributes

This is a 32-bit register. See [CTI register summary](#) for more information.

The following figure shows the CTI\_APPCLR bit assignments.

Figure 20-5: CTI\_APPCLR bit assignments



The following table describes the CTI\_APPCLR bit assignments.

Table 20-7: CTI\_APPCLR bit assignments

Field	Name	Type	Description
[31:4]	Reserved	-	<b>RES0</b>
[3:0]	APPCLEAR	RW	<p>Clears the corresponding internal channel flag.</p> <p><b>0</b> This value has no effect. <b>1</b> This value clears the channel output.</p> <p>The reset value is 0b0000.</p>

## 20.7 CTI\_APPPULSE, CTI Application Channel Pulse Register

The CTI\_APPPULSE register allows software to pulse any channel output. Software can use this register to pulse a channel event in place of a hardware source on a trigger input.

### Usage constraints

Access is only allowed from privileged code. Unprivileged access results in a BusFault being raised.

### Configurations

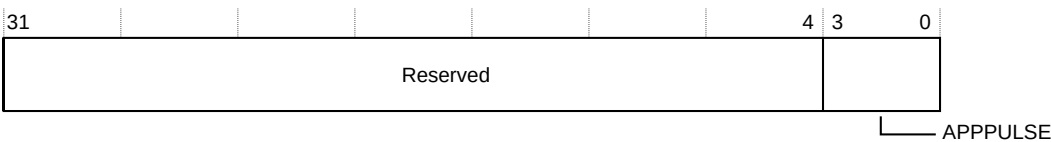
This register is always implemented when the CTI is included.

### Attributes

This is a 32-bit register. See [CTI register summary](#) for more information.

The following figure shows the CTI\_APPPULSE bit assignments.

Figure 20-6: CTI\_APPPULSE bit assignments



The following table describes the CTI\_APPPULSE bit assignments.

Table 20-8: CTI\_APPPULSE bit assignments

Field	Name	Type	Description
[31:4]	Reserved	-	RES0
[3:0]	APPULSE	WO	Pulses the channel outputs.  0 This value has no effect. 1 Pulse channel event for one clock cycle.

## 20.8 CTI\_INEN<n>, n=0-5, CTI Trigger <n> to Channel Enable Register

The CTI\_INEN<n> registers map trigger inputs to channels in the cross trigger system.

### Usage constraints

Access is only allowed from privileged code. Unprivileged access results in a BusFault being raised.

## Configurations

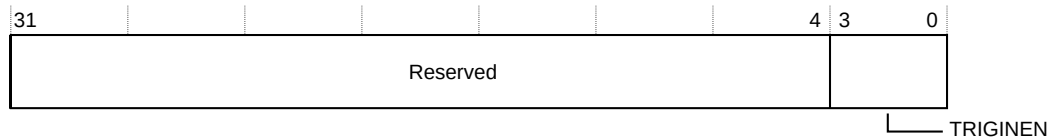
This register is always implemented when the CTI is included.

## Attributes

These are 32-bit registers. See [CTI register summary](#) for more information.

The following figure shows the CTI\_INEN<n> bit assignments, where n=0-5.

**Figure 20-7: CTI\_INEN<n> bit assignments, where n=0-5**



The following table describes the CTI\_INEN<n> bit assignments, where n=0-5.

**Table 20-9: CTI\_INEN<n> bit assignments, where n=0-5**

Field	Name	Type	Description
[31:4]	Reserved	-	RES0
[3:0]	TRIGINEN	RW	<p>Trigger input to channel mapping.</p> <p><b>0</b> Input trigger events are ignored by the corresponding channel.</p> <p><b>1</b> When an event is received on CTITRIGIN, an event is generated on the channel corresponding to this bit.</p> <p>The reset value is 0b0000.</p>

The following table provides more information on CTITRIGIN bit mapping.

**Table 20-10: Cortex®-M52 processor CTI input trigger signals assignment**

Signal	Description	Connection	Acknowledge, handshake
CTITRIGIN[7]	Unused	ETM to CTI <b>Note:</b> If the ETM is not included, bits [4] and [5] are unused and tied LOW.	Pulsed
CTITRIGIN[6]	Unused		
CTITRIGIN[5]	ETM Event Output 1		
CTITRIGIN[4]	ETM Event Output 0 or DWT Comparator Output 3		
CTITRIGIN[3]	DWT Comparator Output 2	Processor to CTI	
CTITRIGIN[2]	DWT Comparator Output 1		
CTITRIGIN[1]	DWT Comparator Output 0		
CTITRIGIN[0]	Processor halted		



## 20.9 CTI\_OUTEN<n>, n=0-7, CTI Channel <n> to Trigger Enable Register

The CTI\_OUTEN<n> registers map trigger outputs to channels in the cross trigger system.

### Usage constraints

Access is only allowed from privileged code. Unprivileged access results in a BusFault being raised.

### Configurations

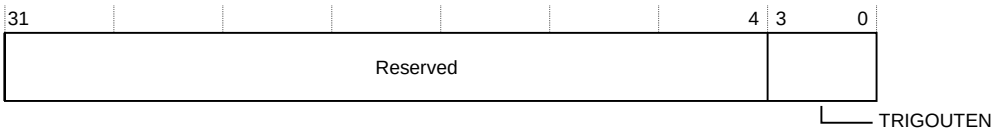
This register is always implemented when the CTI is included.

### Attributes

These are 32-bit registers. See [CTI register summary](#) for more information.

The following figure shows the CTI\_OUTEN<n> bit assignments, where n=0-7.

Figure 20-8: CTI\_OUTEN<n> bit assignments, where n=0-7



The following table describes the CTI\_OUTEN<n> bit assignments, where n=0-7.

Table 20-11: CTI\_OUTEN<n> bit assignments, where n=0-7

Field	Name	Type	Description
[31:4]	Reserved	-	RES0
[3:0]	TRIGOUTEN	RW	<p>Channel to trigger enable mapping.</p> <p><b>0</b> The corresponding channel is ignored by the output triggers.</p> <p><b>1</b> When an event occurs on the channel corresponding to this bit, an event is generated on CTITRIGOUT.</p> <p>The reset value is 0b0000.</p>

The following table provides more information on CTITRIGOUT bit mapping.



**Table 20-13: CTI\_TRIGINSTATUS bit assignments**

Field	Name	Type	Description
[31:8]	Reserved	-	RES0
[7:0]	TRIGINSTATUS	RO	<p>Trigger input status. One bit per trigger.</p> <p><b>0</b> Input is LOW. <b>1</b> Input is HIGH.</p> <p>The reset value is <b>UNKNOWN</b>.</p>

## 20.11 CTI\_TRIGOUTSTATUS, CTI Trigger Output Status Register

The CTI\_TRIGOUTSTATUS register provides the trigger output status.

### Usage constraints

Access is only allowed from privileged code. Unprivileged access results in a BusFault being raised.

### Configurations

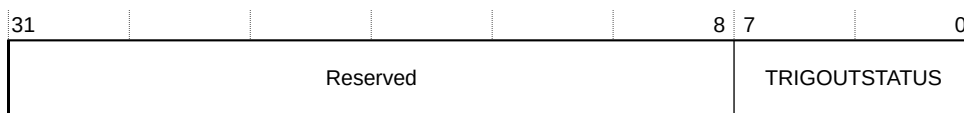
This register is always implemented when the CTI is included.

### Attributes

This is a 32-bit register. See [CTI register summary](#) for more information.

The following figure shows the CTI\_TRIGOUTSTATUS bit assignments.

**Figure 20-10: CTI\_TRIGOUTSTATUS bit assignments**



The following table describes the CTI\_TRIGOUTSTATUS bit assignments.

**Table 20-14: CTI\_TRIGOUTSTATUS bit assignments**

Field	Name	Type	Description
[31:8]	Reserved	-	RES0

Field	Name	Type	Description
[7:0]	TRIGOUTSTATUS	RO	<p>Trigger output status. One bit per trigger.</p> <p><b>0</b> Output is LOW. <b>1</b> Output is HIGH.</p> <p>The reset value is <b>UNKNOWN</b>.</p>

## 20.12 CTI\_CHINSTATUS, CTI Channel Input Status Register

The CTI\_CHINSTATUS register provides the channel input status.

### Usage constraints

Access is only allowed from privileged code. Unprivileged access results in a BusFault being raised.

### Configurations

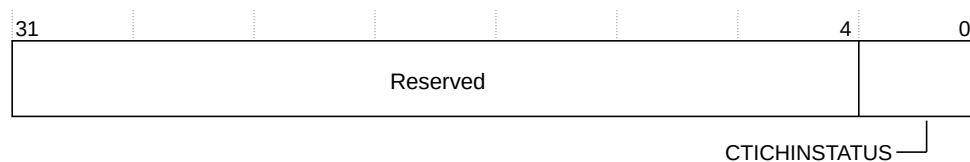
This register is always implemented when the CTI is included.

### Attributes

This is a 32-bit register. See [CTI register summary](#) for more information.

The following figure shows the CTI\_CHINSTATUS bit assignments.

**Figure 20-11: CTI\_CHINSTATUS bit assignments**



The following table describes the CTI\_CHINSTATUS bit assignments.

**Table 20-15: CTI\_CHINSTATUS bit assignments**

Field	Name	Type	Description
[31:4]	Reserved	-	<b>RES0</b>
[3:0]	CTI_CHINSTATUS	RO	<p>Channel input status. One bit per channel input.</p> <p><b>0</b> Input is LOW. <b>1</b> Input is HIGH.</p> <p>The reset value is <b>UNKNOWN</b>.</p>

## 20.13 CTI\_CHOUTSTATUS, CTI Channel Output Status Register

The CTI\_CHOUTSTATUS register provides the channel output status.

### Usage constraints

Access is only allowed from privileged code. Unprivileged access results in a BusFault being raised.

### Configurations

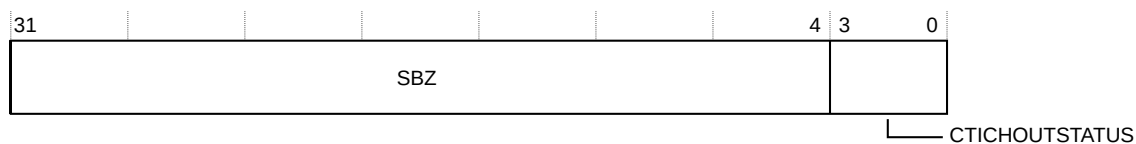
This register is always implemented when the CTI is included.

### Attributes

This is a 32-bit register. See [CTI register summary](#) for more information.

The following figure shows the CTI\_CHOUTSTATUS bit assignments.

**Figure 20-12: CTI\_CHOUTSTATUS bit assignments**



The following table describes the CTI\_CHOUTSTATUS bit assignments.

**Table 20-16: CTI\_CHOUTSTATUS bit assignments**

Field	Name	Type	Description
[31:4]	-	-	Reserved, <b>RES0</b> .
[3:0]	CTI_CHOUTSTATUS	RO	<p>Channel output status. One bit per channel output.</p> <p><b>0</b> Output is LOW. <b>1</b> Output is HIGH.</p> <p>The reset value is <b>UNKNOWN</b>.</p>

## 20.14 CTI\_CHANNELGATE, CTI Channel Gate Register

The CTI\_CHANNELGATE register is the channel output gate.

### Usage constraints

Access is only allowed from privileged code. Unprivileged access results in a BusFault being raised.

### Configurations

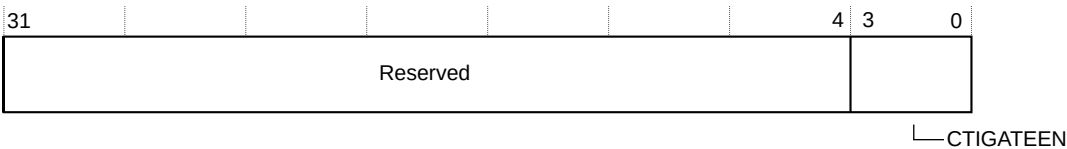
This register is always implemented when the CTI is included.

Attributes

This is a 32-bit register. See [CTI register summary](#) for more information.

The following figure shows the CTI\_CHANNELGATE bit assignments.

Figure 20-13: CTI\_CHANNELGATE bit assignments



The following table describes the CTI\_CHANNELGATE bit assignments.

Table 20-17: CTI\_CHANNELGATE bit assignments

Field	Name	Type	Description
[31:4]	Reserved	-	RES0
[3:0]	CTIGATEEN	RW	Enables the propagation of channel events out of the CTI. Propagation occurs one bit per channel.  <b>0</b> Disable a channel from propagating. <b>1</b> Enable channel propagation.  The reset value is 0b1111.

20.15 CTI\_ITCHOUT, Integration Test Channel Output Register

The CTI\_ITCHOUT register is used to generate channel events.

Usage constraints

Access is only allowed from privileged code. Unprivileged access results in a BusFault being raised.

Configurations

This register is always implemented when the CTI is included.

Attributes

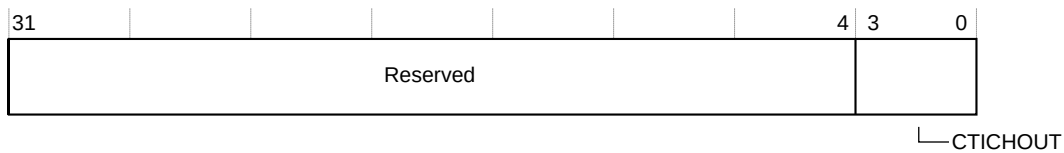
This is a 32-bit register. See [CTI register summary](#) for more information.



Writes to CTI\_ITCHOUT and CTI\_ITTRIGOUT only take effect when integration test mode is enabled using CTI\_ITCONTROL.IME. For more information on CTI\_ITCONTROL, see [CTI\\_ITCONTROL, Integration Mode Control Register](#).

The following figure shows the CTI\_ITCHOUT bit assignments.

**Figure 20-14: CTI\_ITCHOUT bit assignments**



The following table describes the CTI\_ITCHOUT bit assignments.

**Table 20-18: CTI\_ITCHOUT bit assignments**

Field	Name	Type	Description
[31:4]	Reserved	-	RES0
[3:0]	CTI_CHOUT	WO	Pulses the channel outputs. <b>0</b> No effect. <b>1</b> Pulse channel event for one CLKIN cycle.

## 20.16 CTI\_ITTRIGOUT, Integration Test Trigger Output Register

The CTI\_ITTRIGOUT register is used to generate trigger events.

### Usage constraints

Access is only allowed from privileged code. Unprivileged access results in a BusFault being raised.

### Configurations

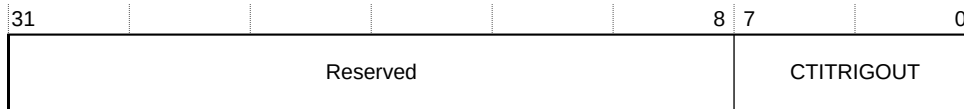
This register is always implemented when the CTI is included.

### Attributes

This is a 32-bit register. See [CTI register summary](#) for more information.

The following figure shows the CTI\_ITTRIGOUT bit assignments.

**Figure 20-15: CTI\_ITTRIGOUT bit assignments**



The following table describes the CTI\_ITTRIGOUT bit assignments.

**Table 20-19: CTI\_ITTRIGOUT bit assignments**

Field	Name	Type	Description
[31:8]	Reserved	-	<b>RES0</b>
[7:0]	CTITRIGOUT	WO	Set/clear trigger output signal. Some output triggers use a software handshake (CTITRIGOUT[3:0]), and others are pulsed (CTITRIGOUT[7:4]).

The following table provides more information on CTITRIGOUT bit mapping.

**Table 20-20: Cortex®-M52 processor CTI output trigger signals assignment**

Signal	Description	Connection	Acknowledge, handshake
CTITRIGOUT[7]	ETM Event Input 3	CTI to ETM <b>Note:</b> If the ETM is not included, bits[7:4] are unused and the output is left untied.	Pulsed
CTITRIGOUT[6]	ETM Event Input 2		
CTITRIGOUT[5]	ETM Event Input 1		
CTITRIGOUT[4]	ETM Event Input 0		
CTITRIGOUT[3]	Interrupt Request 1	CTI to system	Acknowledged by software writing to CTIINTACK register in the interrupt service routine.
CTITRIGOUT[2]	Interrupt Request 0		
CTITRIGOUT[1]	Processor Restart Request	CTI to processor	Processor restarted
CTITRIGOUT[0]	Processor Debug Halt Request		Acknowledged by the debugger writing to the CTIINTACK register.

## 20.17 CTI\_ITCHIN, Integration Test Channel Input Register

The CTI\_ITCHIN register is used to view channel events. The integration test register includes a latch that is set when a pulse is received on a channel input. When read, a register bit reads as 1 if the channel has received a pulse since it was last read. The act of reading the register automatically clears the 1 to 0. When performing integration testing it is therefore important to coordinate the setting of event latches and reading/clearing them.



Usage constraints

Access is only allowed from privileged code. Unprivileged access results in a BusFault being raised.

Configurations

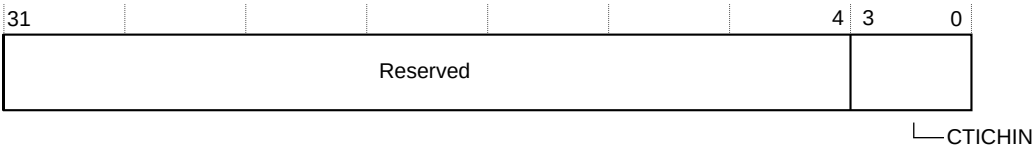
This register is always implemented when the CTI is included.

Attributes

This is a 32-bit register. See [CTI register summary](#) for more information.

The following figure shows the CTI\_ITCHIN bit assignments.

Figure 20-16: CTI\_ITCHIN bit assignments



The following table describes the CTI\_ITCHIN bit assignments.

Table 20-21: CTI\_ITCHIN bit assignments

Field	Name	Type	Description
[31:4]	Reserved	-	RES0
[3:0]	CTICHIN	RO	Reads the latched value of the channel inputs. The reset value is 0b0000.

## 20.18 CTI\_ITTRIGIN, Integration Test Trigger Input Register

The CTI\_ITTRIGIN register is used to view trigger events.

Usage constraints

Access is only allowed from privileged code. Unprivileged access results in a BusFault being raised.

Configurations

This register is always implemented when the CTI is included.

Attributes

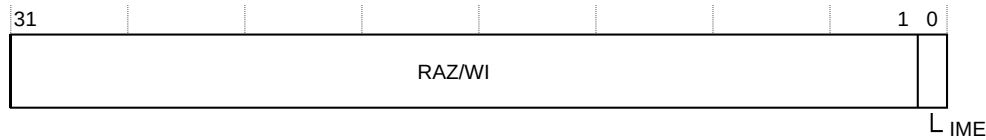
This is a 32-bit register. See [CTI register summary](#) for more information.

The following figure shows the CTI\_ITTRIGIN bit assignments.



The following figure shows the CTI\_ITCONTROL bit assignments.

**Figure 20-18: CTI\_ITCONTROL bit assignments**



The following table describes the CTI\_ITCONTROL bit assignments.

**Table 20-24: CTI\_ITCONTROL bit assignments**

Field	Name	Type	Description
[31:1]	RAZ/WI	-	Read-As-Zero, Writes Ignored.
[0]	IME	RW	Integration Mode Enable. When set, the component enters integration mode, enabling topology detection or integration testing to be performed. The reset value is 0b0.

## 20.20 CTI\_DEVARCH, Device Architecture Register

The CTI\_DEVARCH register identifies the architect and architecture of the CoreSight™ Cross Trigger Interface (CTI).

### Usage constraints

Access is only allowed from privileged code. Unprivileged access results in a BusFault being raised.

### Configurations

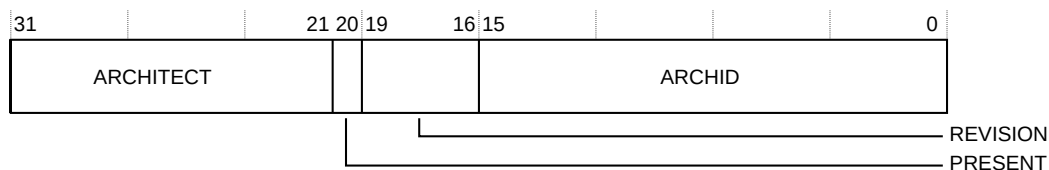
This register is always implemented when the CTI is included.

### Attributes

This is a 32-bit register. See [CTI register summary](#) for more information.

The following figure shows the CTI\_DEVARCH bit assignments.

**Figure 20-19: CTI\_DEVARCH bit assignments**



The following table describes the CTI\_DEVARCH bit assignments.

**Table 20-25: CTI\_DEVARCH bit assignments**

Field	Name	Type	Description
[31:21]	ARCHITECT	RO	Defines the architect of the CTI.  [31:28] Indicates the JEP106 continuation code. [27:21] Indicates the JEP106 identification code.  Arm® is the architect, therefore, this field is 0x575.
[20]	PRESENT	RO	Indicates the presence of this register. This field returns 0x1.
[19:16]	REVISION	RO	Architecture revision. This field returns 0x0000.
[15:0]	ARCHID	RO	Architecture ID. This field returns a value of 0x1A14, indicating the CoreSight™ CTI architecture, version 3.0.

## 20.21 CTI\_DEVID, Device Configuration Register

The CTI\_DEVID register indicates the capability of the *Cross Trigger Interface* (CTI).

### Usage constraints

Access is only allowed from privileged code. Unprivileged access results in a BusFault being raised.

### Configurations

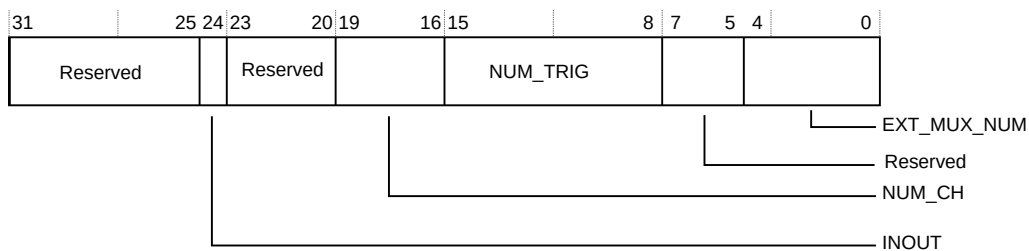
This register is always implemented when the CTI is included.

### Attributes

This is a 32-bit register. See [CTI register summary](#) for more information.

The following figure shows the CTI\_DEVID bit assignments.

**Figure 20-20: CTI\_DEVID bit assignments**



The following table describes the CTI\_DEVID bit assignments.

**Table 20-26: CTI\_DEVID bit assignments**

Field	Name	Type	Description
[31:25]	Reserved	-	RES0.



## 20.23 CTI\_PIDR4, Peripheral Identification Register 4

The CTI\_PIDR4 register provides information about the memory size and JEP106 continuation code that the CoreSight™ *Cross Trigger Interface* (CTI) component uses.

### Usage constraints

Access is only allowed from privileged code. Unprivileged access results in a BusFault being raised.

### Configurations

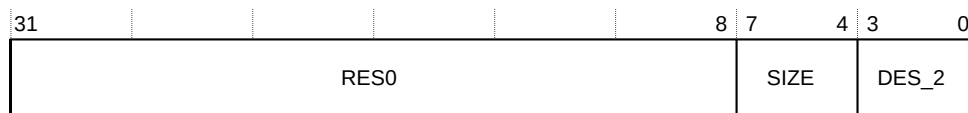
This register is always implemented when the CTI is included.

### Attributes

This is a 32-bit register. See [CTI register summary](#) for more information.

The following figure shows the CTI\_PIDR4 bit assignments.

**Figure 20-22: CTI\_PIDR4 bit assignments**



The following table describes the CTI\_PIDR4 bit assignments.

**Table 20-28: CTI\_PIDR4 bit assignments**

Field	Name	Type	Description
[31:8]	Reserved	-	RES0.
[7:4]	SIZE	RO	This field indicates the memory size that the CTI uses. This field returns 0x0 indicating that the component uses an <b>UNKNOWN</b> number of 4KB blocks. The reset value of this field is 0x0.
[3:0]	DES_2	RO	JEP106 continuation code. Together with CTI_PIDR2.DES_1 and CTI_PIDR1.DES_0, they indicate the designer of the component, not the implementer, except where the two are the same.  The reset value of this field is 0xA.

## 20.24 CTI\_PIDR5, Peripheral Identification Register 5

The CTI\_PIDR5 register is reserved.

### Usage constraints

Access is only allowed from privileged code. Unprivileged access results in a BusFault being raised.

### Configurations

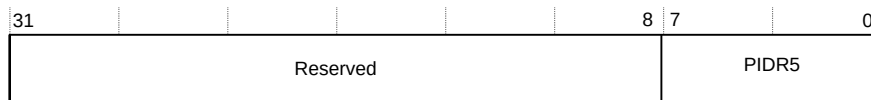
This register is always implemented when the CTI is included.

## Attributes

This is a 32-bit register. See [CTI register summary](#) for more information.

The following figure shows the CTI\_PIDR5 bit assignments.

**Figure 20-23: CTI\_PIDR5 bit assignments**



The following table describes the CTI\_PIDR5 bit assignments.

### Table 20-29: CTI\_PIDR5 bit assignments

Field	Name	Type	Description
[31:8]	Reserved	-	RES0.
[7:0]	PIDR5	RO	RES0.

## 20.25 CTI\_PIDR6, Peripheral Identification Register 6

The CTI\_PIDR6 register is reserved.

## Usage constraints

Access is only allowed from privileged code. Unprivileged access results in a BusFault being raised.

## Configurations

This register is always implemented when the CTI is included.

## Attributes

This is a 32-bit register. See [CTI register summary](#) for more information.

The following figure shows the CTI\_PIDR6 bit assignments.

**Figure 20-24: CTI\_PIDR6 bit assignments**



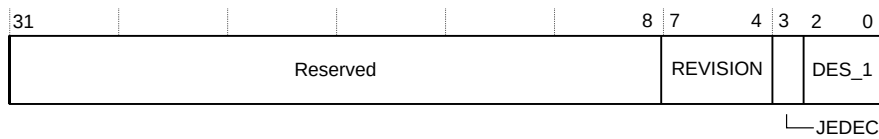








### Figure 20-28: CTI\_PIDR2 bit assignments



The following table describes the CTI\_PIDR2 bit assignments.

### Table 20-34: CTI\_PIDR2 bit assignments

Field	Name	Type	Description
[31:8]	Reserved	-	<b>RES0.</b>
[7:4]	REVISION	RO	This field indicates the revision number of the CTI component. It is an incremental value starting at 0x0 for the first design. The reset value is 0x0.
[3]	JEDEC	RO	This field is always 1, indicating that a JEDEC assigned value is used.
[2:0]	DES_1	RO	This field is the JEP106 identification code, bits[6:4]. Together, with CTI_PIDR4.DES_2 and CTI_PIDR1.DES_0, they indicate the designer of the component and not the implementer, except where the two are the same. The reset value is 0b111.

## 20.30 CTI\_PIDR3, Peripheral Identification Register 3

The CTI\_PIDR3 register indicates minor errata fixes of the *Cross Trigger Interface* (CTI) component and if you have modified the behavior of the component.

## Usage constraints

Access is only allowed from privileged code. Unprivileged access results in a BusFault being raised.

## Configurations

This register is always implemented when the CTI is included.

## Attributes

This is a 32-bit register. See [CTI register summary](#) for more information.

The following figure shows the CTI PIDR3 bit assignments.

**Figure 20-29: CTI\_PIDR3 bit assignments**



The following table describes the CTI\_PIDR3 bit assignments.

**Table 20-35: CTI\_PIDR3 bit assignments**

Field	Name	Type	Description
[31:8]	Reserved	-	RES0.
[7:4]	REVAND	RO	This field indicates minor errata fixes specific to this design, for example metal fixes after implementation. This field is 0x0 without ECO.
[3:0]	CMOD	RO	Customer modified. Where the component is reusable IP, this value indicates whether you have modified the behavior of the component. This field is 0x0 without ECO.

## 20.31 CTI\_CIDR0, Component Identification Register 0

The CTI\_CIDR0 register indicates the preamble.

### Usage constraints

Access is only allowed from privileged code. Unprivileged access results in a BusFault being raised.

### Configurations

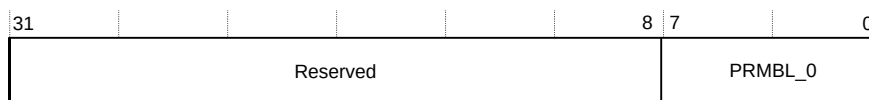
This register is always implemented when the CTI is included.

### Attributes

This is a 32-bit register. See [CTI register summary](#) for more information.

The following figure shows the CTI\_CIDR0 bit assignments.

**Figure 20-30: CTI\_CIDR0 bit assignments**



The following table describes the CTI\_CIDR0 bit assignments.

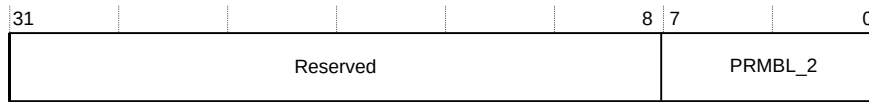
**Table 20-36: CTI\_CIDR0 bit assignments**

Field	Name	Type	Description
[31:8]	Reserved	-	RES0.
[7:0]	PRMBL_0	RO	Preamble. This field returns 0x0D.



The following figure shows the CTI\_CIDR2 bit assignments.

**Figure 20-32: CTI\_CIDR2 bit assignments**



The following table describes the CTI\_CIDR2 bit assignments.

### Table 20-38: CTI\_CIDR2 bit assignments

Field	Name	Type	Description
[31:8]	Reserved	-	<b>RES0.</b>
[7:0]	PRMBL_2	RO	Preamble. This field returns 0x05.

### 20.34 CTI\_CIDR3, Component Identification Register 3

The CTI\_CIDR3 register indicates the preamble.

## Usage constraints

Access is only allowed from privileged code. Unprivileged access results in a BusFault being raised.

## Configurations

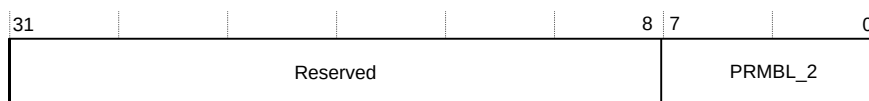
This register is always implemented when the CTI is included.

## Attributes

This is a 32-bit register. See [CTI register summary](#) for more information.

The following figure shows the CTI\_CIDR3 bit assignments.

**Figure 20-33: CTI\_CIDR3 bit assignments**



The following table describes the CTI\_CIDR3 bit assignments.

**Table 20-39: CTI\_CIDR3 bit assignments**

Field	Name	Type	Description
[31:8]	Reserved	-	<b>RES0.</b>
[7:0]	PRMBL_3	RO	Preamble. This field returns 0xB1.

# 21. BreakPoint Unit

This chapter describes the *BreakPoint Unit* (BPU).

## 21.1 BPU features

The *BreakPoint Unit* (BPU) is an implementation of the architectural *Flash Patch and Breakpoint* (FPB) unit. The BPU can be configured with four or eight instruction address comparators. Each comparator supports breakpoint functionality on all instructions that are fetched across the entire address range in which code is located.

The BPU does not support flash patching. Flash patching allows a small programmable memory in the system to apply patches to program memory that cannot be modified.

The BPU functionality is largely architecturally defined. The **IMPLEMENTATION DEFINED** functionality includes:

### Security

If the Cortex®-M52 processor is configured to include the Security Extension and if invasive debug is not enabled for the security mode that the processor was in when the breakpoint became active, then debug events that are associated with breakpoints are blocked.

### Architectural remap registers

The Cortex®-M52 processor does not include the address remapping functionality for instructions and literals. Therefore, the following architecturally defined registers have the following behavior:

- FP\_REMAP.RMPSPT is RAZ/WI.
- FP\_REMAP.REMAP is Reserved.
- FP\_CTRL.NUM\_LIT is 0, indicating that no literal comparators are included.
- Attempting to enable Flash Patch in FP\_COMPn is ignored.

Also, only instruction address comparators are supported.

For more information on the registers listed in this section, see the *Arm®v8-M Architecture Reference Manual*.

## 21.2 BPU register summary

The following table shows the *BreakPoint Unit* (BPU) registers, with address, name, type and reset information for each register.

Depending on the implementation of your processor, some of these registers might not be present. Any register that is configured as not present reads as zero and ignores writes.



All BPU registers are described in the *Arm®v8-M Architecture Reference Manual*.

**Table 21-1: BPU register summary**

Address	Name	Type	Reset value	Description
0xE0002000	FP_CTRL	RW	<ul style="list-style-type: none"> <li>If four instruction comparators are implemented, the reset value is 0x10000040.</li> <li>If eight instruction comparators are implemented, the reset value is 0x10000080.</li> </ul>	Flash Patch Control Register
0xE0002004	FP_REMAP	RAZ/WI	-	Flash Patch Remap. This register is not implemented.
0xE0002008	FP_COMP0	RW	0x00000000	Flash Patch Comparator Register 0-7 <b>Note:</b> <ul style="list-style-type: none"> <li>FP_COMPn[0] is reset to 0.</li> <li>FP_COMPn[31:1] is reset to <b>UNKNOWN</b>.</li> <li>If only 4 breakpoints are implemented, FP_COMP4-FP_COMP7 are RAZ/WI.</li> </ul>
0xE000200C	FP_COMP1	RW		
0xE0002010	FP_COMP2	RW		
0xE0002014	FP_COMP3	RW		
0xE0002018	FP_COMP4	RW		
0xE000201C	FP_COMP5	RW		
0xE0002020	FP_COMP6	RW		
0xE0002024	FP_COMP7	RW		
0xE0002FBC	FP_DEVARCH	RO	0x47701A03	FPB CoreSight™ Device Architecture Register
0xE0002FD0	FP_PIDR4	RO	0x0000000A	Peripheral identification Register 4
0xE0002FE0	FP_PIDR0	RO	0x00000024	Peripheral identification Register 0
0xE0002FE4	FP_PIDR1	RO	0x0000005D	Peripheral identification Register 1
0xE0002FE8	FP_PIDR2	RO	0x0000000F	Peripheral identification Register 2
0xE0002FEC	FP_PIDR3	RO	0x00000000	Peripheral identification Register 3
0xE0002FF0	FP_CIDR0	RO	0x0000000D	Component identification registers
0xE0002FF4	FP_CIDR1	RO	0x00000090	
0xE0002FF8	FP_CIDR2	RO	0x00000005	
0xE0002FFC	FP_CIDR3	RO	0x000000B1	



FP\_DEVTYPE, FP\_PIDR5, FP\_PIDR6, and FP\_PIDR7 registers are not implemented, and are **RES0**.

# Appendix A External Wakeup Interrupt Controller

This appendix describes the *External Wakeup Interrupt Controller* (EWIC) that can be used with the Cortex®-M52 processor.

## A.1 EWIC features

The Cortex®-M52 processor supports the *External Wakeup Interrupt Controller* (EWIC), which is a peripheral to the processor and is suitable for sleep states when it is the only source of wakeup in the system. The EWIC stores state to allow the processor to wake up from retention or powered off state.

An APB interface controls the EWIC which must be connected to the External Private Peripheral Bus (EPPB) interface of the processor. This interface is used to communicate all interrupt and event status information on sleep entry and wakeup. The EWIC interface can be asynchronous to the processor by instantiating an asynchronous clock domain crossing in the system on the APB interface.

### EWIC configuration

The EWIC can be configured to support a variable number of events.

A minimum of 4 events are supported:

- External event
- Debug request
- Non-Maskable Interrupt, NMI
- One interrupt

A maximum of 483 events are supported:

- External event
- Debug request
- NMI
- 480 interrupts

Any number of events in the range 4-483 is permitted.



The EWIC can support fewer interrupts than the processor supports. Interrupts above those that the EWIC supports cannot cause the core to exit low-power state. Therefore, higher numbered interrupts that occur when the core is in a low-power state might be lost.

## A.2 EWIC register summary

The *External Wakeup Interrupt Controller* (EWIC) requires memory-mapped registers that are accessed at address 0xE0047000 onwards in the PPB region of the memory map. The registers are contained in a CoreSight™ compliant 4KB block. The following table shows the EWIC registers.

**Table A-1: EWIC register summary**

Address	Name	Type	Reset value	Description
0xE0047000	EWIC_CR	RW	0x00000000	EWIC_CR, EWIC Control Register
0xE0047004	EWIC_ASCR	RW	0x00000003	EWIC_ASCR, EWIC Automatic Sequence Control Register
0xE0047008	EWIC_CLRMASK	WO	0x00000000	EWIC_CLRMASK, EWIC Clear Mask Register
0xE004700C	EWIC_NUMID	RO	0x0000XXXX	EWIC_NUMID, EWIC Event Number ID Register
0xE0047200	EWIC_MASKA	RW	0x0000000X	EWIC_MASKA and EWIC_MASKn, EWIC Mask Registers
0xE0047204 - 0xE004723C	EWIC_MASKn	RW	UNKNOWN	
0xE0047400	EWIC_PENDA	RO	0x0000000X	EWIC_PENDA and EWIC_PENDn, EWIC Pend Event Registers
0xE0047404 - 0xE004743C	EWIC_PENDn	RW	UNKNOWN	
0xE0047600	EWIC_PSR	RO	0x0000XXXX	EWIC_PSR, EWIC Pend Summary Register
0xE0047604 - 0xE0047EFC	-	UNK/ SBZP	-	Reserved
0xE0047F00 - 0xE0047FFC	CoreSight™ registers	RO	-	EWIC register summary

### A.2.1 EWIC\_CR, EWIC Control Register

The EWIC\_CR is the main *External Wakeup Interrupt Controller* (EWIC) control register.

#### Usage constraints

When the EWIC is connected to the *External Private Peripheral Bus* (EPPB) interface, the Cortex®-M52 processor controls access to these registers using the following constraints:

- If the Security Extension is included, then access from Non-secure software is only allowed if AIRCR.BFHFNMINS is set to 1.
- Access is only allowed from privileged code. Unprivileged access results in a BusFault being raised.

#### Configurations

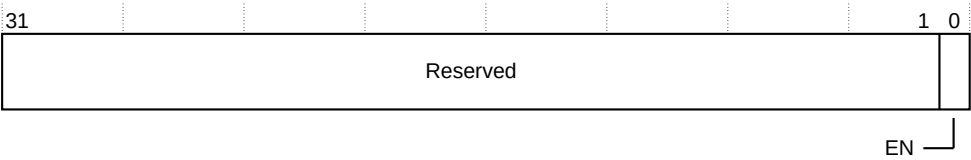
This register is always implemented when the EWIC is included.

Attributes

This is a 32-bit register. See [EWIC register summary](#) for more information.

The following figure shows the EWIC\_CR bit assignments.

Figure A-1: EWIC\_CR bit assignments



The following table describes the EWIC\_CR bit assignments.

Table A-2: EWIC\_CR bit assignments

Field	Name	Type	Description
[31:1]	-	-	Reserved, <b>RES0</b>
[0]	EN	RW	<p>The options are:</p> <p><b>0</b> EWIC is disabled, events are not pended, and WAKEUP is not signaled.</p> <p><b>1</b> EWIC is enabled, events are pended, and WAKEUP is signaled.</p> <p>The reset value is 0.</p>

A.2.2 EWIC\_ASCR, EWIC Automatic Sequence Control Register

The EWIC\_ASCR determines whether the processor generates APB transactions on entry and exit from *Wakeup Interrupt Controller* (WIC) sleep to set up the wakeup state in the *External Wakeup Interrupt Controller* (EWIC).

Usage constraints

- When the EWIC is connected to the *External Private Peripheral Bus* (EPPB) interface, the Cortex®-M52 processor controls access to these registers using the following constraints:
- If the Security Extension is included, then access from Non-secure software is only allowed if AIRCR.BFHFNMINs is set to 1.
  - Access is only allowed from privileged code. Unprivileged access results in a BusFault being raised.

Configurations

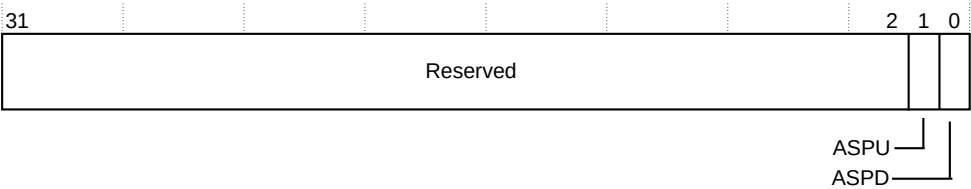
This register is always implemented when the EWIC is included.

Attributes

This is a 32-bit register. See [EWIC register summary](#) for more information.

The following figure shows the EWIC\_ASCR bit assignments.

Figure A-2: EWIC\_ASCR bit assignments



The following table describes the EWIC\_ASCR bit assignments.

Table A-3: EWIC\_ASCR bit assignments

Field	Name	Type	Description
[31:2]	-	-	Reserved, <b>RESO</b>
[1]	ASPU	RW	<p>The value of this bit is sent to the processor. The processor must use this value to decide whether any automatic EWIC accesses must be performed on transitioning from a low-power state. The options are:</p> <p><b>0</b> No automatic sequence on powerup.</p> <p><b>1</b> Automatic sequence on powerup.</p> <p>The reset value is 1.</p>
[0]	ASPD	RW	<p>The value of this bit is sent to the processor. The processor must use this value to decide whether any automatic EWIC accesses must be performed on transitioning to a low-power state. The options are:</p> <p><b>0</b> No automatic sequence on entry to a low-power state.</p> <p><b>1</b> Automatic sequence on entry to a low-power state.</p> <p>The reset value is 1.</p>



Note

- If the automatic sequence is disabled, then software can program the unit by writing to the EWIC\_MASKA and EWIC\_MASKn registers on sleep entry and reading from the EWIC\_PENDn registers on sleep exit. For more information, see [EWIC\\_MASKA and EWIC\\_MASKn, EWIC Mask Registers](#) and [EWIC\\_PENDA and EWIC\\_PENDn, EWIC Pend Event Registers](#).
- The value of EWIC\_ASCR does not affect the operation of the EWIC itself. It only affects the control information that is driven on the WICCONTROL signal to the Cortex®-M52 processor.
- When modifying EWIC\_ASCR.ASPU and EWIC\_ASCR.ASPD, the resulting changes to WICCONTROL[3:0] must be stable before software enters sleep and remain stable until software execution resumes. Otherwise, modification of these registers can result in **UNPREDICTABLE** behavior.

### A.2.3 EWIC\_CLRMASK, EWIC Clear Mask Register

When there is a write to the EWIC\_CLRMASK register, it causes EWIC\_MASKA and all the EWIC\_MASKn registers to be cleared. The write data is ignored. This register is RAZ.

### A.2.4 EWIC\_NUMID, EWIC Event Number ID Register

The EWIC\_NUMID register returns the total number of events that are supported in the *External Wakeup Interrupt Controller* (EWIC).

#### Usage constraints

When the EWIC is connected to the *External Private Peripheral Bus* (EPPB) interface, the Cortex®-M52 processor controls access to these registers using the following constraints:

- If the Security Extension is included, then access from Non-secure software is only allowed if AIRCR.BFHFNMINS is set to 1.
- Access is only allowed from privileged code. Unprivileged access results in a BusFault being raised.

#### Configurations

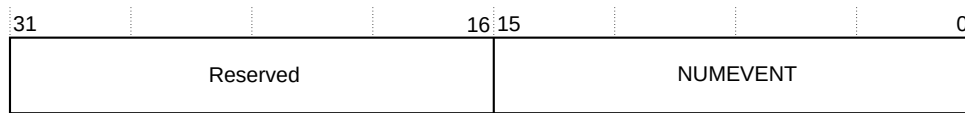
This register is always implemented when the EWIC is included.

#### Attributes

This is a 32-bit register. See [EWIC register summary](#) for more information.

The following figure shows the EWIC\_NUMID bit assignments.

### Figure A-3: EWIC\_NUMID bit assignments



The following table describes the EWIC\_NUMID bit assignments.

### Table A-4: EWIC\_NUMID bit assignments

Field	Name	Type	Description
[31:16]	-	-	Reserved, <b>RES0</b>
[15:0]	NUMEVENT	RO	The number of events supported.

### A.2.5 EWIC\_MASKA and EWIC\_MASKn, EWIC Mask Registers

The EWIC\_MASKA register defines the mask for special events and the EWIC\_MASKn registers for external interrupt (IRQ) events. There is one EWIC\_MASKn register implemented for every 32 external interrupts that the *External Wakeup Interrupt Controller* (EWIC) supports. At least one register is always implemented. EWIC\_MASKn is at address 0x0047204+(n×4), where n=0-14.

## Usage constraints

When the EWIC is connected to the *External Private Peripheral Bus* (EPPB) interface, the Cortex®-M52 processor controls access to these registers using the following constraints:

- If the Security Extension is included, then access from Non-secure software is only allowed if AIRCR.BFHFNMINS is set to 1.
- Access is only allowed from privileged code. Unprivileged access results in a BusFault being raised.

## Configurations

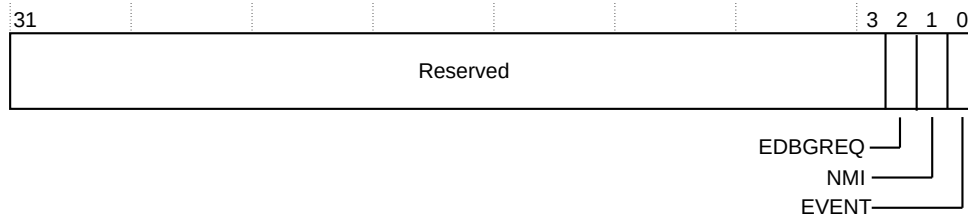
These registers are always implemented when the EWIC is included.

## Attributes

These are 32-bit registers. See [EWIC register summary](#) for more information.

The following figure shows the EWIC MASKA bit assignments.

**Figure A-4: EWIC\_MASKA bit assignments**



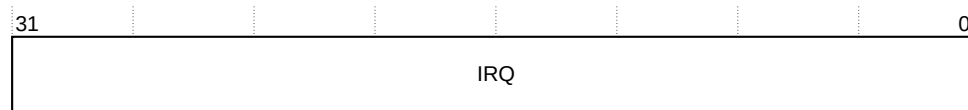
The following table describes the EWIC\_MASKA bit assignments.

**Table A-5: EWIC\_MASKA bit assignments**

Field	Name	Type	Description
[31:3]	-	-	Reserved, <b>RES0</b>
[2]	EDBGREQ	RW	Mask for external debug request. If this bit is 0, the mask is enabled.
[1]	NMI	RW	Mask for Non-Maskable Interrupt, NMI. If this bit is 0, the mask is enabled.
[0]	EVENT	RW	Mask for <i>Wait For Exception</i> (WFE) wakeup event. If this bit is 0, the mask is enabled.

The following figure shows the EWIC\_MASKn, where n=0-14, bit assignments.

**Figure A-5: EWIC\_MASKn, where n=0-14 bit assignments**



The following table describes the EWIC\_MASKn, where n=0-14, bit assignments.

**Table A-6: EWIC\_MASKn, where n=0-14, bit assignments**

Field	Name	Type	Description
[31:0]	IRQ	RW	Masks for external interrupts (n×32) to ((n+1)×32)-1. If any of the bits are 0, the mask is enabled for the associated interrupt. Additionally, any interrupt that the WIC does not support is also RAZ.

## A.2.6 EWIC\_PENDA and EWIC\_PENDn, EWIC Pend Event Registers

These registers indicate which events have been pended. The EWIC\_PENDA register is used for special events and the EWIC\_PENDn registers are used for external interrupt (IRQ) events. There is



one EWIC\_PENDn register implemented for each 32 external interrupt events the EWIC supports. EWIC\_PENDA and at least one EWIC\_PENDn register is always implemented.

### Usage constraints

When the EWIC is connected to the *External Private Peripheral Bus* (EPPB) interface, the Cortex®-M52 processor controls access to these registers using the following constraints:

- If the Security Extension is included, then access from Non-secure software is only allowed if AIRCR.BFHFNMINS is set to 1.
- Access is only allowed from privileged code. Unprivileged access results in a BusFault being raised.

### Configurations

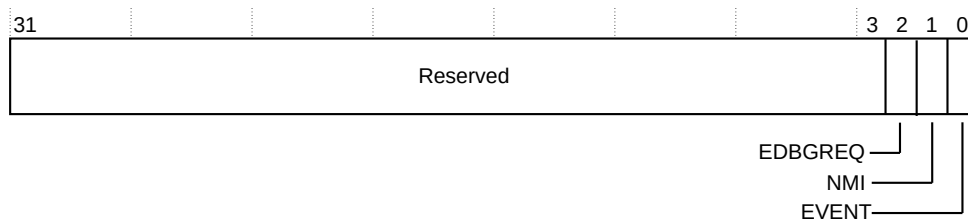
These registers are always implemented when the EWIC is included. There is one EWIC\_PENDn register implemented for every 32 events that the *External Wakeup Interrupt Controller* (EWIC) supports. At least one register is always implemented. EWIC\_MASKn is at address  $0xE0047404 + (n \times 4)$ .

### Attributes

These are 32-bit registers. The EWIC\_PENDn registers can be written to transfer pended interrupts in the NVIC when the processor enters sleep. EWIC\_PENDA is read-only as special events can only be pended by the system (usually during sleep). See [EWIC register summary](#) for more information.

The following figure shows the EWIC\_PENDA bit assignments.

**Figure A-6: EWIC\_PENDA bit assignments**



The following table describes the EWIC\_PENDA bit assignments.

**Table A-7: EWIC\_PENDA bit assignments**

Field	Name	Type	Description
[31:3]	-	-	Reserved, <b>RES0</b>
[2]	EDBGREQ	RO	External debug request is pended.
[1]	NMI	RO	Non-Maskable Interrupt, NMI, is pended.
[0]	EVENT	RO	<i>Wait For Exception</i> (WFE) wakeup event is pended.

The following figure shows the EWIC\_PENDn, where n=0-14, bit assignments.

**Figure A-7: EWIC\_PENDn, where n=0-14 bit assignments**



The following table describes the EWIC\_PENDn, where n=0-14, bit assignments.

**Table A-8: EWIC\_PENDn, where n=0-14, bit assignments**

Field	Name	Type	Description
[31:0]	IRQ	RW	Interrupts (n×32) to ((n+1)×32)-1 are pending. A write of zero to this field is ignored.



Any IRQ bit associated with an interrupt that the EWIC does not support is RAZ/WI. All EWIC\_PENDn registers are reset 0. If an event occurs when EWIC\_CR.EN is set, then the corresponding bit in EWIC\_PENDn is set. All EWIC\_PENDn registers are cleared if the EWIC is disabled, that is, if EWIC\_CR.EN is cleared. For more information on EWIC\_CR, see [EWIC\\_CR, EWIC Control Register](#).

## A.2.7 EWIC\_PSR, EWIC Pend Summary Register

The EWIC\_PSR indicates which EWIC\_PENDn registers are nonzero. This allows the processor to efficiently determine which EWIC\_PENDn registers need to be read. This can be used to improve code efficiency in the powerup sequence.

### Usage constraints

When the EWIC is connected to the *External Private Peripheral Bus* (EPPB) interface, the Cortex®-M52 processor controls access to these registers using the following constraints:

- If the Security Extension is included, then access from Non-secure software is only allowed if AIRCR.BFHFNMINS is set to 1.
- Access is only allowed from privileged code. Unprivileged access results in a BusFault being raised.

### Configurations

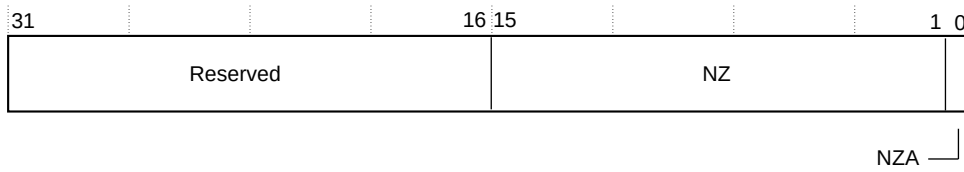
This register is always implemented when the EWIC is included.

### Attributes

This is a 32-bit register. See [EWIC register summary](#) for more information.

The following figure shows the EWIC\_PSR bit assignments.

**Figure A-8: EWIC\_PSR bit assignments**



The following table describes the EWIC\_PSR bit assignments.

**Table A-9: EWIC\_PSR bit assignments**

Field	Name	Type	Description
[31:16]	-	-	Reserved, <b>RES0</b>
[15:1]	NZ	RO	If EWIC_PSR.NZ[n+1] is set, then EWIC_PENDn is nonzero.
[0]	NZA	RO	If EWIC_PSR.NZA set, then EWIC_PENDA is nonzero.



If any bit of EWIC\_PSR is associated with an EWIC\_PENDn register that is entirely RAZ/WI, then the bit in EWIC\_PSR is also RAZ/WI.

## A.2.8 EWIC CoreSight™ register summary

The *External Wakeup Interrupt Controller* (EWIC) implements the standard CoreSight™ registers.

The following table describes the CoreSight™ registers that the EWIC implements.

**Table A-10: EWIC CoreSight™ register summary**

Address	Name	Type	Reset value	Description
0xE0047F00	EWIC_ITCTRL	RO	0x00000000	Integration Mode Control Register
0xE0047F04-0xE0047F9C	-	-	-	Reserved
0xE0047FA0	EWIC_CLAIMSET	RW	0x0000000F	Claim Tag Set Register
0xE0047FA4	EWIC_CLAIMCLR	RW	0x00000000	Claim Tag Clear Register
0xE0047FA8	EWIC_DEVAFF0	RO	0x80000000	Device Affinity Register 0
0xE0047FAC	EWIC_DEVAFF1	RO	0x00000000	Device Affinity Register 1
0xE0047FB0	EWIC_LAR	WO	<b>UNKNOWN</b>	Lock Access Register
0xE0047FB4	EWIC_LSR	RO	0x00000000	Lock Status Register
0xE0047FB8	EWIC_AUTHSTATUS	RO	0x00000000	Authentication Status Register
0xE0047FBC	EWIC_DEVARCH	RO	0x47700A07	Device Architecture Register
0xE0047FC0	EWIC_DEVID2	RO	0x00000000	Device Configuration Register 2
0xE0047FC4	EWIC_DEVID1	RO	0x00000000	Device Configuration Register 1

Address	Name	Type	Reset value	Description
0xE0047FC8	EWIC_DEVID	RO	0x00000000	Device Configuration Register
0xE0047FCC	EWIC_DEVTYPE	RO	0x00000000	Device Type Identifier Register
0xE0047FD0	EWIC_PIDR4	RO	0x0000000A	Peripheral Identification Registers
0xE0047FD4	EWIC_PIDR5	RO	0x00000000	
0xE0047FD8	EWIC_PIDR6	RO	0x00000000	
0xE0047FDC	EWIC_PIDR7	RO	0x00000000	
0xE0047FE0	EWIC_PIDR0	RO	0x00000024	
0xE0047FE4	EWIC_PIDR1	RO	0x0000005D	
0xE0047FE8	EWIC_PIDR2	RO	0x0000000F	
0xE0047FEC	EWIC_PIDR3	RO	0x00000000	
0xE0047FF0	EWIC_CIDR0	RO	0x0000000D	Component Identification Registers
0xE0047FF4	EWIC_CIDR1	RO	0x00000090	
0xE0047FF8	EWIC_CIDR2	RO	0x00000005	
0xE0047FFC	EWIC_CIDR3	RO	0x000000B1	



For more information on these registers, see the *Arm® CoreSight™ Architecture Specification v3.0*. In the *Arm® CoreSight™ Architecture Specification v3.0*, these register names are not prefixed with "EWIC\_".

## A.2.9 EWIC\_CLAIMSET, EWIC Claim Tag Set Register

The EWIC\_CLAIMSET register is used to set whether functionality is in use by a debug agent. All debug agents must implement a common protocol to use these bits.

For more information on example protocols, see the *Arm® CoreSight™ Architecture Specification v3.0*.

### Usage constraints

See [EWIC CoreSight register summary](#) for more information.

### Configurations

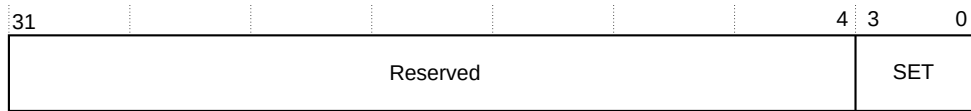
This register is always implemented.

### Attributes

This is a 32-bit register.

The following figure shows the EWIC\_CLAIMSET bit assignments.

### Figure A-9: EWIC\_CLAIMSET bit assignments



The following table describes the EWIC\_CLAIMSET bit assignments.

### Table A-11: EWIC\_CLAIMSET bit assignments

Field	Name	Type	Description
[31:4]	Reserved	-	RES0
[3:0]	SET	RW	<p>The options are:</p> <p><b>Write 0</b> No effect.</p> <p><b>Write 1</b> Set the claim tag for bit[n].</p> <p><b>Read 0</b> The claim tag that is represented by bit[n] is not implemented.</p> <p><b>Read 1</b> The claim tag that is represented by bit[n] is implemented.</p>

#### A.2.10 EWIC\_CLAIMCLR, EWIC Claim Tag Clear Register

The EWIC\_CLAIMCLR register is used to set whether functionality is in use by a debug agent. All debug agents must implement a common protocol to use these bits.

For more information on example protocols, see the *Arm® CoreSight™ Architecture Specification v3.0*.

## Usage constraints

See [EWIC CoreSight register summary](#) for more information.

## Configurations

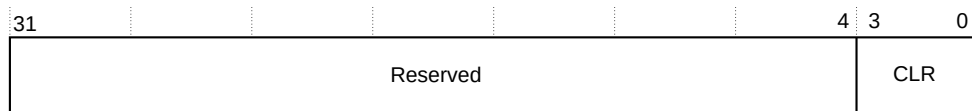
This register is always implemented.

## Attributes

This is a 32-bit register.

The following figure shows the EWIC CLAIMCLR bit assignments.

**Figure A-10: EWIC\_CLAIMCLR bit assignments**



The following table describes the EWIC\_CLAIMCLR bit assignments.

**Table A-12: EWIC\_CLAIMCLR bit assignments**

Field	Name	Type	Description
[31:4]	Reserved	-	RES0
[3:0]	CLR	RW	<p>The options are:</p> <p><b>Write 0</b> No effect.</p> <p><b>Write 1</b> Clear the claim tag for bit[n].</p> <p><b>Read 0</b> The claim tag that is represented by bit[n] is not set.</p> <p><b>Read 1</b> The claim tag that is represented by bit[n] is set.</p>

# Appendix B Trace Port Interface Unit

This appendix describes the *Trace Port Interface Unit* (TPIU) that can be used with the Cortex®-M52 processor.

## B.1 TPIU features

The Cortex®-M52 *Trace Port Interface Unit* (TPIU) is an optional component that bridges between the on-chip trace data from the *Embedded Trace Macrocell* (ETM) and the *Instrumentation Trace Macrocell* (ITM), with separate IDs, to a data stream.

The Cortex®-M52 TPIU encapsulates IDs where required, and an external *Trace Port Analyzer* (TPA) captures the data stream.

The Cortex®-M52 TPIU is specially designed for low-cost debug. If your implementation requires additional debugging features, the following options are available:

- CoreSight™ TPIU-M, see the *Arm® CoreSight™ TPIU-M Technical Reference Manual* for more information
- CoreSight™ SoC-600 TPIU, see the *Arm® CoreSight™ System-on-Chip SoC-600 Technical Reference Manual* for more information



In this chapter, the term TPIU refers to the Cortex®-M52 processor TPIU.

The *Trace Port Interface Unit* (TPIU) supports up to two ATB ports. The following table shows the various ATB1 and ATB2 parameters configuration options.

**Table B-1: ATB port parameters**

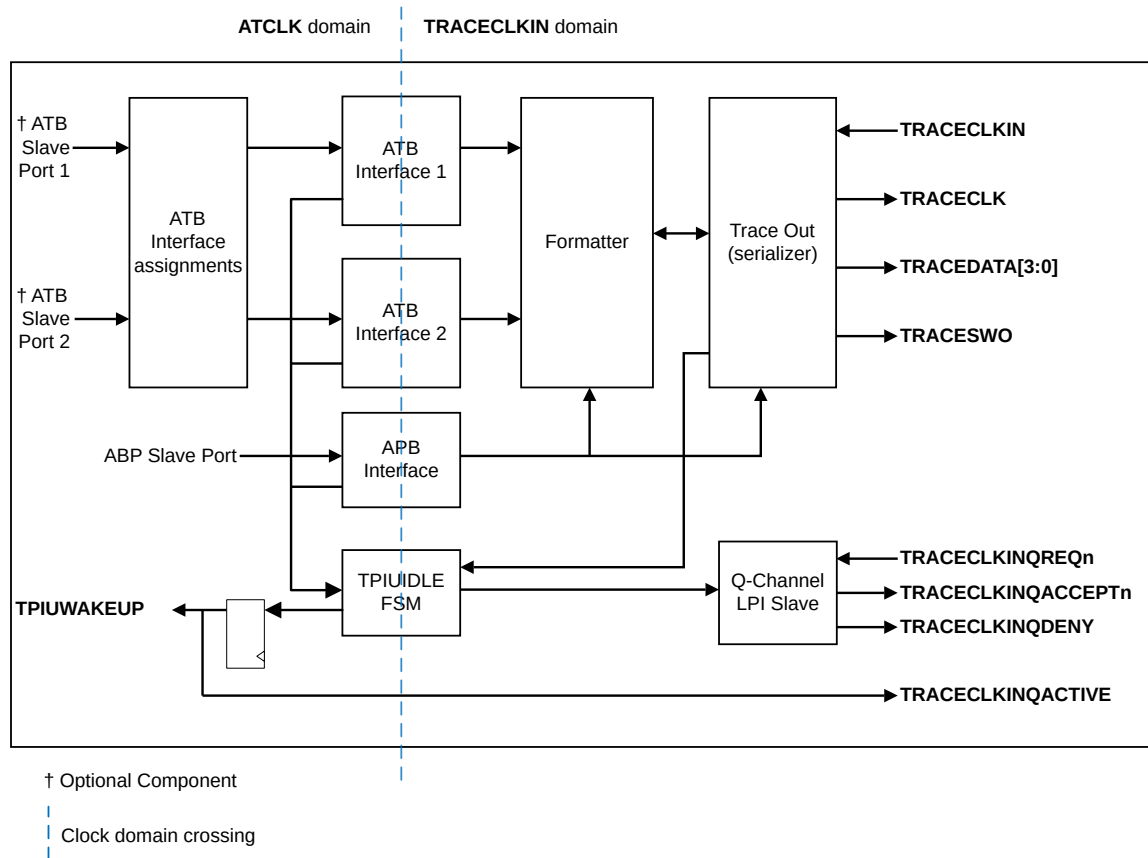
ATB1	ATB2	Description
0	0	Illegal combination. If the ITM and ETM do not exist, then the TPIU is not present.
0	1	ATB port 2 is present, and Arm® recommends connecting the ETM to it. In this case, the ATB interface 2 logic is removed and gets assigned to ATB interface 1 logic.
1	0	ATB port 1 is present, and Arm® recommends connecting the ITM to it.
1	1	Both ports are present, and Arm® recommends that the ITM is connected to ATB port 1 and the ETM is connected to ATB port 2.



If your system design uses the optional ETM component, the TPIU configuration supports both ITM and ETM debug trace. See the *Arm China CoreSight™ ETM-M52 Technical Reference Manual*.

The following figure shows the component layout of the TPIU when  $\text{ATB1}$  and  $\text{ATB2}$  are set to 1.

**Figure B-1: TPIU block diagram**



### B.1.1 TPIU Formatter

The formatter inserts source ID signals into the data packet stream so that trace data can be re-associated with its trace source. The formatter is always active when the Trace Port Mode is active.

The formatting protocol is described in the *Arm® CoreSight™ Architecture Specification v3.0*. You must enable synchronization in the DWT or TPIU\_PSCR to provide synchronization for the formatter.

When the formatter is enabled, if there is no data to output after a frame has been started, half-sync packets can be inserted. Distributed synchronization from the DWT or TPIU\_PSCR causes synchronization which ensures that any partial frame is completed, and at least one full synchronization packet is generated.



## B.1.2 Serial Wire Output format

The TPIU can output trace data in a *Serial Wire Output* (SWO) format:

- TPIU\_DEVID specifies the formats that are supported. See [TPIU\\_DEVID, Device Configuration Register](#)
- TPIU\_SPPR specifies the SWO format in use. See the *Arm®v8-M Architecture Reference Manual*.

When one of the two SWO modes is selected, you can enable the TPIU to bypass the formatter for trace output. When the formatter is bypassed, only data on the ATB interface 1 is passed through and ATB interface 2 data is discarded.



When operating in bypass mode, Arm® recommends that in a configuration that supports an ETM and ITM, the ITM data is passed through by connecting the ITM to the ATB Slave Port 1.

## B.2 TPIU register summary

The following table shows the *Trace Port Interface Unit* (TPIU) registers. Depending on the implementation of your processor, the TPIU registers might not be present, and the CoreSight™ TPIU might be present instead. Any register that is configured as not present reads as zero.



Arm® recommends reprogramming the TPIU before any data has been presented on either ATB slave port and after either of the following:

- Both ATRESETn and TRESETn have been applied
- A flush has been completed using FFCR.FOnMan.

If this recommendation is not followed, reprogramming can cause either momentary or permanent data corruption that might require ATRESETn and TRESETn to be applied. This corruption is related to trace and not general data corruption of execution state or memory.

**Table B-2: TPIU IMPLEMENTATION DEFINED register summary**

Address	Name	Type	Reset	Description
0xE0040000	TPIU_SSPSR	RO	- <b>Note:</b> The value at reset corresponds to the MAXPORTSIZE configuration tie off.	TPIU_SSPSR, Supported Port Size Register
0xE0040004	TPIU_CSPSR	RW	0x00000001	TPIU_CSPSR, Current Port Size Register

Address	Name	Type	Reset	Description
0xE0040010	TPIU_ACPR	RW	0x00000000	TPIU_ACPR, Asynchronous Clock Prescaler Register
0xE00400F0	TPIU_SPPR	RW	0x00000001	TPIU_SPPR, Selected Pin Protocol Register
0xE0040300	TPIU_FFSR	RO	0x00000008	TPIU_FFSR, Formatter and Flush Status Register
0xE0040304	TPIU_FFCR	RW	0x00000102	TPIU_FFCR, Formatter and Flush Control Register
0xE0040308	TPIU_PSCR	RW	0x00000000	TPIU_PSCR, Periodic Synchronization Counter Register
0xE0040EE8	TPIU_TRIGGER	RO	0x00000000	TPIU_TRIGGER, TPIU TRIGGER Register
0xE0040EEC	TPIU_ITFTTD0	RO	UNKNOWN	ITFTTD0, Integration Test FIFO Test Data 0 Register
0xE0040EF0	TPIU_ITATBCTR2	RW	0x00000000	ITATBCTR2, Integration Test ATB Control Register 2
0xE0040EF8	TPIU_ITATBCTR0	RO	0x00000000	ITATBCTR0, Integration Test ATB Control 0 Register
0xE0040EFC	TPIU_ITFTTD1	RO	UNKNOWN	ITFTTD1, Integration Test FIFO Test Data 1 Register
0xE0040F00	TPIU_ITCTRL	RW	0x00000000	TPIU_ITCTRL, Integration Mode Control
0xE0040FA0	TPIU_CLAIMSET	RW	0x0000000F	CLAIMSET, Claim Tag Set Register
0xE0040FA4	TPIU_CLAIMCLR	RW	0x00000000	CLAIMCLR, Claim Tag Clear Register
0xE0040FC8	TPIU_DEVID	RO	0x00000CA0/0x00000CA1	TPIU_DEVID, Device Configuration Register
0xE0040FCC	TPIU_DEVTYPE	RO	0x00000011	TPIU_DEVTYPE, Device Type Identifier Register
0xE0040FD0	TPIU_PIDR4	RO	0x0000000A	TPIU_PIDR4, Peripheral Identification Register 4
0xE0040FD4	TPIU_PIDR5	RO	0x00000000	TPIU_PIDR5, Peripheral Identification Register 5
0xE0040FD8	TPIU_PIDR6	RO	0x00000000	TPIU_PIDR6, Peripheral Identification Register 6
0xE0040FDC	TPIU_PIDR7	RO	0x00000000	TPIU_PIDR7, Peripheral Identification Register 7
0xE0040FE0	TPIU_PIDR0	RO	0x00000024	TPIU_PIDR0, Peripheral Identification Register 0
0xE0040FE4	TPIU_PIDR1	RO	0x0000005D	TPIU_PIDR1, Peripheral Identification Register 1
0xE0040FE8	TPIU_PIDR2	RO	0x0000000F	TPIU_PIDR2, Peripheral Identification Register 2

Address	Name	Type	Reset	Description
0xE0040FEC	TPIU_PIDR3	RO	0x00000000 <b>Note:</b> The value of TPIU_PIDR3[7:4] is determined by MCU_ECOREVNUM[11:8].	TPIU_PIDR3, Peripheral Identification Register 3
0xE0040FF0	TPIU_CIDR0	RO	0x0000000D	TPIU_CIDR0, Component Identification Register 0
0xE0040FF4	TPIU_CDR1	RO	0x00000090	TPIU_CDR1, Component Identification Register 1
0xE0040FF8	TPIU_CDR2	RO	0x00000005	TPIU_CDR2, Component Identification Register 2
0xE0040FFC	TPIU_CDR3	RO	0x000000B1	TPIU_CDR3, Component Identification Register 3

## B.2.1 TPIU\_SSPSR, Supported Port Size Register

TPIU\_SSPSR shows the supported sizes of the trace data port TRACEDATE[3:0]. Each bit location represents a single port size that is supported, that is, sizes from 32 bits to 1 bit in bit location [31:0]. If a bit is set, then that port size is supported. The supported trace port sizes are limited by the MAXPORTSIZE signal. The maximum possible trace port size for Cortex®-M52 is 4 bits.

For more information on the MAXPORTSIZE signal, see the *Arm® Cortex®-M52 Processor Integration and Implementation Manual*. The *Arm® Cortex®-M52 Processor Integration and Implementation Manual* is a confidential document and available to licensees only and Arm® partners with an NDA agreement.

### Usage constraints

There are no usage constraints.

### Configurations

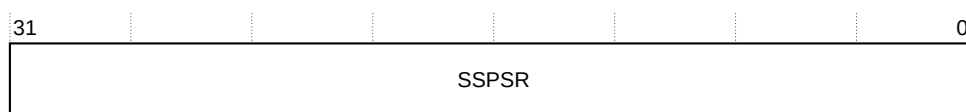
Available in all configurations.

### Attributes

See [TPIU\\_IMPLEMENTATION\\_DEFINED register summary](#).

The following figure shows the TPIU\_SSPSR bit assignments.

**Figure B-2: TPIU\_SSPSR bit assignments**



The following table shows the TPIU\_SSPSR bit assignments.

**Table B-3: TPIU\_SSPSR bit assignments**

Bits	Name	Function
[31:0]	SSPSR	Supported sizes of TRACEDATA[3:0]. The possible values are:  <div> <div>0bh</div> <div>Maximum 1-bit trace port.</div> </div> <div> <div>0b0011</div> <div>Maximum 2-bit trace port.</div> </div> <div> <div>0b1011</div> <div>Maximum 4-bit trace port.</div> </div>

## B.2.2 TPIU\_CSPSR, Current Port Size Register

TPIU\_CSPSR shows the currently selected size of the trace data port, TRACEDATA[3:0].

It has the same format as the TPIU\_SSPSR register, but only one bit is set to show the currently selected port size. If a bit that is indicated as not supported in the TPIU\_SSPSR is set in the TPIU\_CSPSR, it can corrupt the output trace stream, in trace capture mode, and the trace patterns in pattern generation mode. If more than one bit is set, the port size is internally resolved to the highest order set bit. This register must not be modified while the trace port is still active, or without correctly stopping the formatter. If this happens, it can result in data not being aligned to the port width.

### Usage constraints

There are no usage constraints.

### Configurations

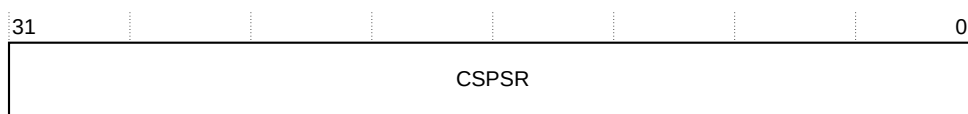
Available in all configurations.

### Attributes

See [TPIU\\_IMPLEMENTATION\\_DEFINED register summary](#).

The following figure shows the TPIU\_CSPSR bit assignments.

**Figure B-3: TPIU\_CSPSR bit assignments**



The following table shows the TPIU\_CSPSR bit assignments.

**Table B-4: TPIU\_CSPSR bit assignments**

Bits	Name	Function
[31:0]	CSPSR	Currently selected size of the trace data port TRACEDATA[3:0]. The possible values are:  <div> <div>0b0001</div> <div>0b0010</div> <div>0b1000</div> </div> <div> <div>1-bit trace port</div> <div>2-bit trace port</div> <div>4-bit trace port</div> </div>

## B.2.3 TPIU\_SPPR, Selected Pin Protocol Register

TPIU\_SPPR selects which protocol is used by the TPIU for trace output.

### Usage constraints

There are no usage constraints.

### Configurations

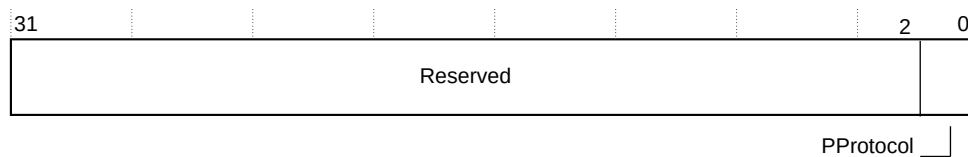
Available in all configurations.

### Attributes

See [TPIU IMPLEMENTATION DEFINED register summary](#).

The following figure shows the TPIU\_SPPR bit assignments.

**Figure B-4: TPIU\_SPPR bit assignments**



The following table shows the TPIU\_SPPR bit assignments.

**Table B-5: TPIU\_SPPR bit assignments**

Bits	Name	Function
[31:2]	-	RES0
[1:0]	PProtocol	Pin protocol used for trace output. The options are:  <div> <div>0x0</div> <div>0x1</div> <div>0x2</div> </div> <div> <div>Parallel port</div> <div>SWO Manchester</div> <div>SWO NRZ (UART)</div> </div>

## B.2.4 TPIU\_PSCR, Periodic Synchronization Counter Register

TPIU\_PSCR determines the reload value of the Periodic Synchronization Counter. This counter enables the frequency of sync packets to be optimized to the trace capture buffer size.

### Usage constraints

There are no usage constraints.

### Configurations

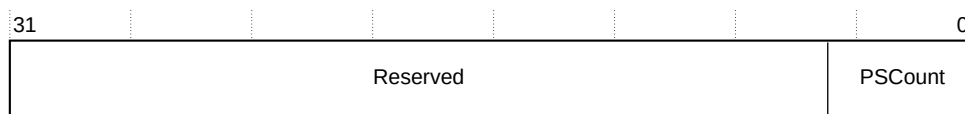
Available in all configurations.

### Attributes

See [TPIU IMPLEMENTATION DEFINED register summary](#).

The following figure shows the TPIU\_PSCR bit assignments.

**Figure B-5: TPIU\_PSCR bit assignments**



The following table shows the TPIU\_PSCR bit assignments.

**Table B-6: TPIU\_PSCR bit assignments**

Bits	Name	Function
[31:5]	-	RAZ/WI
[4:0]	PSCount	<p>Periodic Synchronization Count that determines the reload value of the Synchronization Counter. The Periodic Synchronization Counter counts up to a maximum of <math>2^{16}</math> bytes, where the TPIU_PSCR.PSCount value determines the reload value of Synchronization Counter, as 2 to the power of the programmed value.</p> <p>The TPIU_PSCR.PSCount value has a range between 0b00111 and 0b10000, any attempt to program register with a value smaller than the minimum value disables the Synchronization Counter. If the programmed reload value is greater than the maximum value, then the Periodic Synchronization Counter is reloaded with its maximum value and the TPIU will generate synchronization requests at this interval.</p>

## B.2.5 TPIU\_ACPR, Asynchronous Clock Prescaler Register

TPIU\_ACPR scales the Baud rate of the asynchronous output.

### Usage constraints

There are no usage constraints.

### Configurations

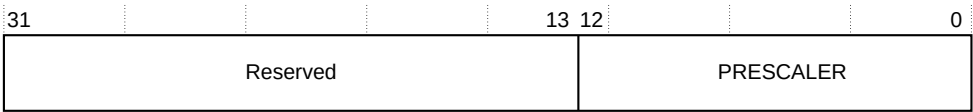
Available in all configurations.

### Attributes

See [TPIU IMPLEMENTATION DEFINED register summary](#).

The following figure shows the TPIU\_ACPR bit assignments.

Figure B-6: TPIU\_ACPR bit assignments



The following table shows the TPIU\_ACPR bit assignments.

Table B-7: TPIU\_ACPR bit assignments

Bits	Name	Function
[31:13]	-	Reserved. RAZ/SBZP.
[12:0]	PRESCALER	Divisor for TRACECLKIN is Prescaler + 1.

B.2.6 TPIU\_FFSR, Formatter and Flush Status Register

TPIU\_FFSR indicates the status of the TPIU formatter.

Usage constraints

There are no usage constraints.

Configurations

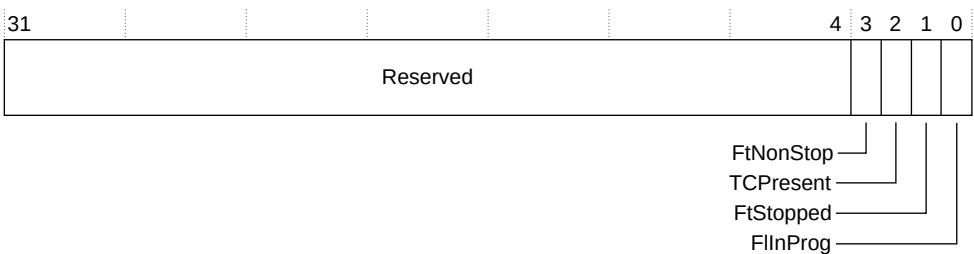
Available in all configurations.

Attributes

See [TPIU IMPLEMENTATION DEFINED register summary](#).

The following figure shows the TPIU\_FFSR bit assignments.

Figure B-7: TPIU\_FFSR bit assignments



The following table shows the TPIU\_FFSR bit assignments.

**Table B-8: TPIU\_FFSR bit assignments**

Bit	Name	Type	Description
[31:4]	Reserved	-	RES0
[3]	FtNonStop	RO	Formatter cannot be stopped
[2]	TCPresent	RO	This bit is always 0b0.
[1]	FtStopped	RO	This bit is always 0b0.
[0]	FIInProg	RO	Flush in progress. The values read can be:  <b>0</b> When all the data received, before the flush is acknowledged, has been output on the trace port <b>1</b> When a flush is initiated

## B.2.7 TPIU\_FFCR, Formatter and Flush Control Register

TPIU\_FFCR controls the TPIU formatter.

### Usage constraints

There are no usage constraints.

### Configurations

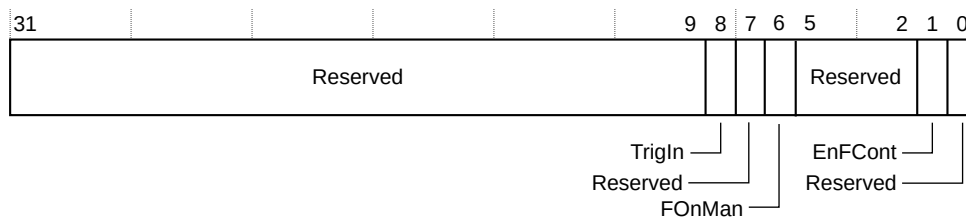
Available in all configurations.

### Attributes

See [TPIU IMPLEMENTATION DEFINED register summary](#).

The following figure shows the TPIU\_FFCR bit assignments.

**Figure B-8: TPIU\_FFCR bit assignments**



The following table shows the TPIU\_FFCR bit assignments.

**Table B-9: TPIU\_FFCR bit assignments**

Bit	Name	Type	Description
[31:9]	Reserved	-	RES0
[8]	TrgIn	-	This bit Reads-As-One (RAO), specifying that triggers are inserted when a trigger pin is asserted.
[7]	Reserved	-	RES0



Bit	Name	Type	Description
[6]	FOnMan	RW	Flush on manual. The options are:  <b>0</b> When the flush completes. Set to 0 on a reset of the TPIU. <b>1</b> Generates a flush.
[5:2]	Reserved	-	<b>RES0</b>
[1]	EnFCont	RW	Enable continuous formatting. The options are:  <b>0</b> Continuous formatting disabled. <b>1</b> Continuous formatting enabled.
[0]	Reserved	-	<b>RES0</b>

The TPIU can output trace data in a *Serial Wire Output* (SWO) format. See [Serial Wire Output format](#).



If TPIU\_SPPR is set to select Trace Port Mode, the formatter is automatically enabled. If you then select one of the SWO modes, TPIU\_FFCR reverts to its previously programmed value.

## B.2.8 TPIU\_TRIGGER, TPIU\_TRIGGER Register

The TPIU\_TRIGGER register controls the integration test TRIGGER input.

### Usage constraints

There are no usage constraints.

### Configurations

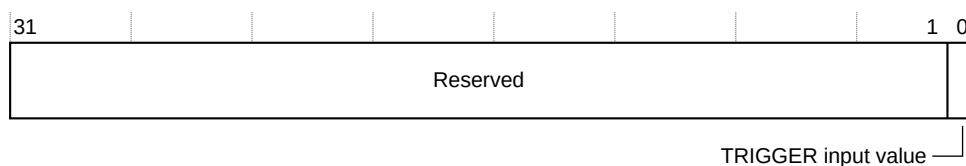
Available in all configurations.

### Attributes

See [TPIU\\_IMPLEMENTATION\\_DEFINED register summary](#).

The following figure shows the TPIU\_TRIGGER bit assignments.

**Figure B-9: TPIU\_TRIGGER bit assignments**



The following table shows the TPIU\_TRIGGER bit assignments.

**Table B-10: TPIU\_TRIGGER bit assignments**

Bit	Name	Type	Description
[31:1]	Reserved	-	RES0
[0]	TRIGGER input value	RO	When read, this bit returns the TRIGGER input value.

## B.2.9 ITFTTDO, Integration Test FIFO Test Data 0 Register

ITFTTDO controls trace data integration testing.

### Usage constraints

You must set bit[1] of TPIU\_ITCTRL to use this register. See [TPIU\\_ITCTRL, Integration Mode Control](#).

### Configurations

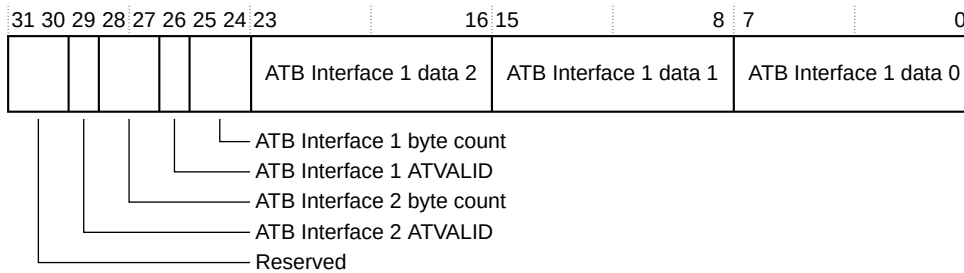
Available in all configurations.

### Attributes

See [TPIU IMPLEMENTATION DEFINED register summary](#).

The following figure shows the Integration Test FIFO Test Data 0 Register data bit assignments.

**Figure B-10: ITFTTDO bit assignments**



The following table shows the ITFTTDO bit assignments.

**Table B-11: ITFTTDO bit assignments**

Bits	Name	Function
[31:30]	-	Reserved.
[29]	ATB Interface 2 ATVALID input	Returns the value of the ATB Interface 2 ATVALID signal.
[28:27]	ATB Interface 2 byte count	Number of bytes of ATB Interface 2 trace data since last read of this register.
[26]	ATB Interface 1 ATVALID input	Returns the value of the ATB Interface 1 ATVALID signal.
[25:24]	ATB Interface 1 byte count	Number of bytes of ATB Interface 1 trace data since last read of this register.
[23:16]	ATB Interface 1 data 2	ATB Interface 1 trace data. The TPIU discards this data when the register is read.
[15:8]	ATB Interface 1 data 1	
[7:0]	ATB Interface 1 data 0	

### B.2.10 ITATBCTR2, Integration Test ATB Control Register 2

ITATBCTR2 controls integration test.

**Usage constraints**

You must set bit[0] of TPIU\_ITCTRL to use this register. See [TPIU\\_ITCTRL, Integration Mode Control](#).

**Configurations**

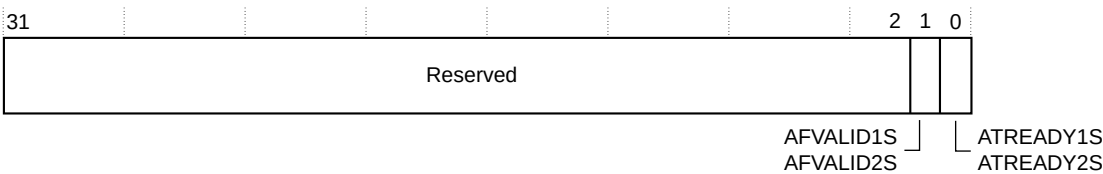
Available in all configurations.

**Attributes**

See [TPIU IMPLEMENTATION DEFINED register summary](#).

The following figure shows the ITATBCTR2 bit assignments.

**Figure B-11: ITATBCTR2 bit assignments**



The following table shows the ITATBCTR2 bit assignments.

**Table B-12: ITATBCTR2 bit assignments**

Bits	Name	Function
[1]	AFVALID1S, AFVALID2S	This bit sets the value of both the ATB Interface 1 and 2 AFVALID outputs, if the TPIU is in integration test mode.
[0]	ATREADY1S, ATREADY2S	This bit sets the value of both the ATB Interface 1 and 2 ATREADY outputs, if the TPIU is in integration test mode.

### B.2.11 ITFTTD1, Integration Test FIFO Test Data 1 Register

ITFTTD1 controls trace data integration testing.

**Usage constraints**

You must set bit[1] of TPIU\_ITCTRL to use this register. See [TPIU\\_ITCTRL, Integration Mode Control](#).

**Configurations**

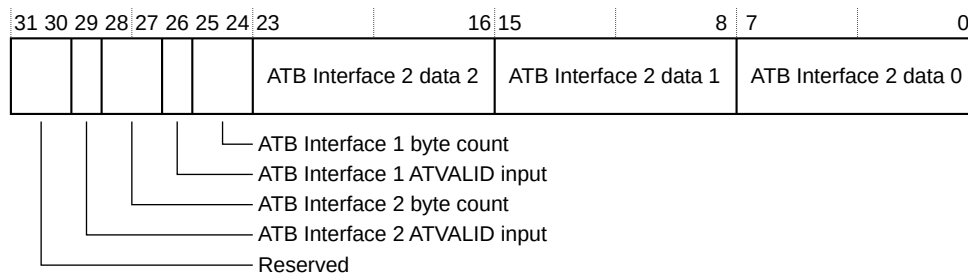
Available in all configurations.

**Attributes**

See [TPIU IMPLEMENTATION DEFINED register summary](#).

The following figure shows the ITFTTD1 bit assignments.

**Figure B-12: ITFTTD1 bit assignments**



The following table shows the ITFTTD1 bit assignments.

**Table B-13: ITFTTD1 bit assignments**

Bits	Name	Function
[31:30]	-	Reserved.
[29]	ATB Interface 2 ATVALID input	Returns the value of the ATB Interface 2 ATVALID signal.
[28:27]	ATB Interface 2 byte count	Number of bytes of ATB Interface 2 trace data since last read of this register.
[26]	ATB Interface 1 ATVALID input	Returns the value of the ATB Interface 1 ATVALID signal.
[25:24]	ATB Interface 1 byte count	Number of bytes of ATB Interface 1 trace data since last read of this register.
[23:16]	ATB Interface 2 data 2	ATB Interface 2 trace data. The TPIU discards this data when the register is read.
[15:8]	ATB Interface 2 data 1	
[7:0]	ATB Interface 2 data 0	

## B.2.12 ITATBCTR0, Integration Test ATB Control 0 Register

ITATBCTR0 is used for integration test.

### Usage constraints

There are no usage constraints.

### Configurations

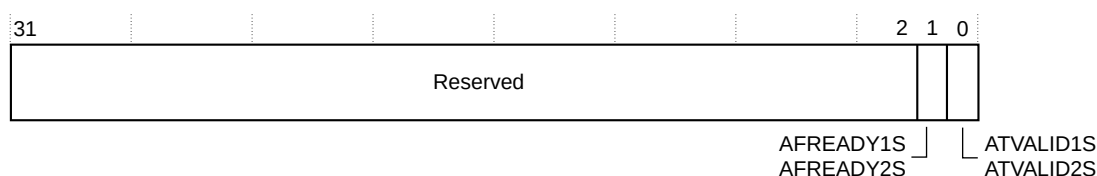
Available in all configurations.

### Attributes

See [TPIU IMPLEMENTATION DEFINED register summary](#).

The following figure shows the ITATBCTR0 bit assignments.

**Figure B-13: ITATBCTR0 bit assignments**



The following table shows the ITATBCTRO bit assignments.

**Table B-14: ITATBCTRO bit assignments**

Bits	Name	Function
[1]	AFREADY1S, AFREADY2S	A read of this bit returns the value of AFREADY1S OR-gated with AFREADY2S.
[0]	ATVALID1S, ATVALID2S	A read of this bit returns the value of ATVALID1S OR-gated with ATVALID2S.

## B.2.13 TPIU\_ITCTRL, Integration Mode Control

TPIU\_ITCTRL specifies normal or integration mode for the TPIU.

### Usage constraints

There are no usage constraints.

### Configurations

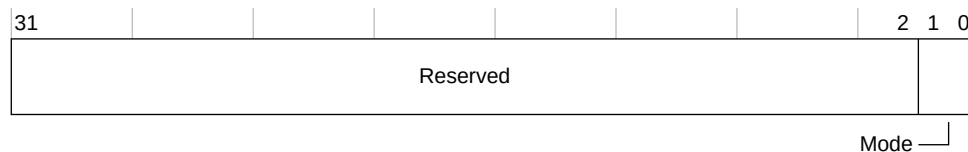
Available in all configurations.

### Attributes

See [TPIU IMPLEMENTATION DEFINED register summary](#).

The following figure shows the TPIU\_ITCTRL bit assignments.

**Figure B-14: TPIU\_ITCTRL bit assignments**



The following table shows the TPIU\_ITCTRL bit assignments.

**Table B-15: TPIU\_ITCTRL bit assignments**

Bits	Name	Function
[31:2]	-	Reserved.
[1:0]	Mode	<p>Specifies the current mode for the TPIU:</p> <p><b>0b00</b> Normal mode.  <b>0b01</b> Integration test mode.  <b>0b10</b> Integration data test mode.  <b>0b11</b> Reserved.</p> <p>In integration data test mode, the trace output is disabled, and data can be read directly from each input port using the integration data registers.</p>

## B.2.14 CLAIMSET, Claim Tag Set Register

The CLAIMSET register is used to set whether functionality is in use by a debug agent. All debug agents must implement a common protocol to use these bits.

For more information on example protocols, see the *Arm® CoreSight™ Architecture Specification v3.0*.

### Usage constraints

See [TPIU register summary](#) for more information.

### Configurations

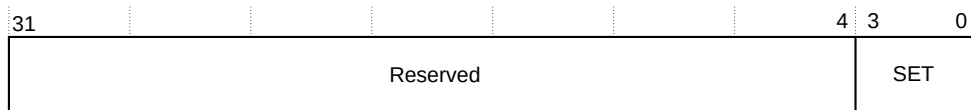
This register is always implemented.

### Attributes

This is a 32-bit register.

The following figure shows the CLAIMSET bit assignments.

**Figure B-15: CLAIMSET bit assignments**



The following table describes the CLAIMSET bit assignments.

**Table B-16: CLAIMSET bit assignments**

Field	Name	Type	Description
[31:4]	Reserved	-	RES0
[3:0]	SET	RW	<p>The options are:</p> <p><b>Write 0</b> No effect.</p> <p><b>Write 1</b> Set the claim tag for bit[n].</p> <p><b>Read 0</b> The claim tag that is represented by bit[n] is not implemented.</p> <p><b>Read 1</b> The claim tag that is represented by bit[n] is implemented.</p>

## B.2.15 CLAIMCLR, Claim Tag Clear Register

The CLAIMCLR register is used to set whether functionality is in use by a debug agent. All debug agents must implement a common protocol to use these bits.

For more information on example protocols, see the *Arm® CoreSight™ Architecture Specification v3.0*.

### Usage constraints

See [TPIU register summary](#) for more information.

### Configurations

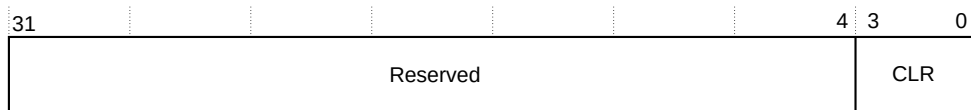
This register is always implemented.

### Attributes

This is a 32-bit register.

The following figure shows the CLAIMCLR bit assignments.

**Figure B-16: CLAIMCLR bit assignments**



The following table describes the CLAIMCLR bit assignments.

**Table B-17: CLAIMCLR bit assignments**

Field	Name	Type	Description
[31:4]	Reserved	-	RES0
[3:0]	CLR	RW	<p>The options are:</p> <p><b>Write 0</b> No effect.</p> <p><b>Write 1</b> Clear the claim tag for bit[n].</p> <p><b>Read 0</b> The claim tag that is represented by bit[n] is not set.</p> <p><b>Read 1</b> The claim tag that is represented by bit[n] is set.</p>

## B.2.16 TPIU\_DEVID, Device Configuration Register

TPIU\_DEVID indicates the functions that are provided by the TPIU for use in the topology detection.

### Usage constraints

There are no usage constraints.

### Configurations

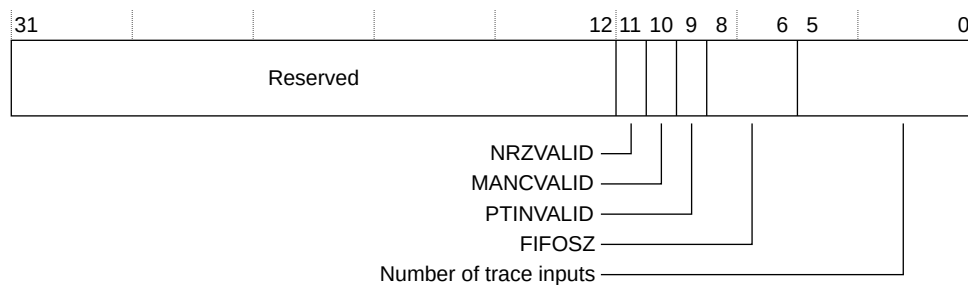
Available in all configurations.

### Attributes

See [TPIU IMPLEMENTATION DEFINED register summary](#).

The following figure shows the TPIU\_DEVID bit assignments.

**Figure B-17: TPIU\_DEVID bit assignments**



The following table shows the TPIU\_DEVID bit assignments.

**Table B-18: TPIU\_DEVID bit assignments**

Bits	Name	Function
[31:12]	-	Reserved.
[11]	NRZVALID	Indicates support for SWO using UART/NRZ encoding.  Always RAO. The output is supported.
[10]	MANCVALID	Indicates support for SWO using Manchester encoding.  Always RAO. The output is supported.
[9]	PTINVALID	Indicates support for parallel trace port operation.  Always RAZ. Trace data and clock modes are supported.
[8:6]	FIFOSZ	Indicates the implemented size of the TPIU output FIFO for trace data:  <b>0b010</b> Four bytes.
[5:0]	Number of trace inputs	Specifies the number of trace inputs:  <b>0b000000</b> One input. <b>0b000001</b> Two inputs.



## B.2.17 TPIU\_DEVTYPE, Device Type Identifier Register

TPIU\_DEVTYPE provides a debugger with information about the component when the Part Number field is not recognized. The debugger can then report this information.

### Usage Constraints

There are no usage constraints.

### Configurations

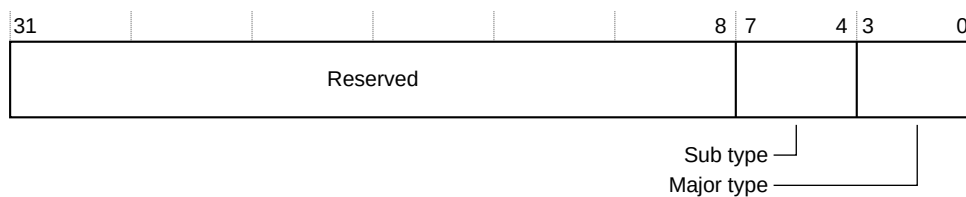
Available in all configurations.

### Attributes

See [TPIU IMPLEMENTATION DEFINED register summary](#).

The following figure shows the TPIU\_DEVTYPE bit assignments.

**Figure B-18: TPIU\_DEVTYPE bit assignments**



The following table shows the TPIU\_DEVTYPE bit assignments.

**Table B-19: TPIU\_DEVTYPE bit assignments**

Bits	Name	Function
[31:8]	-	Reserved.
[7:4]	Sub type	<b>0x1</b> Identifies the classification of the debug component.
[3:0]	Major type	<b>0x1</b> Indicates this device is a trace sink and specifically a TPIU.

## B.2.18 TPIU\_PIDR4, Peripheral Identification Register 4

The TPIU\_PIDR4 register provides information about the memory size and JEP106 continuation code that the *Trace Port Interface Unit* (TPIU) component uses.

### Usage constraints

Access is only allowed from privileged code. Unprivileged access results in a BusFault being raised.

## Configurations

This register is always implemented when the TPIU is included.

## Attributes

This is a 32-bit register. See [CTI register summary](#) for more information.

The following figure shows the TPIU\_PIDR4 bit assignments.

**Figure B-19: TPIU\_PIDR4 bit assignments**



The following table describes the TPIU\_PIDR4 bit assignments.

**Table B-20: TPIU\_PIDR4 bit assignments**

Field	Name	Type	Description
[31:8]	Reserved	-	<b>RES0</b> .
[7:4]	SIZE	RO	This field indicates the memory size that the TPIU uses. This field returns 0x0 indicating that the component uses an <b>UNKNOWN</b> number of 4KB blocks. The reset value of this field is 0x0.
[3:0]	DES_2	RO	JEP106 continuation code. Together with TPIU_PIDR2.DES_1 and TPIU_PIDR1.DES_0, they indicate the designer of the component, not the implementer, except where the two are the same.  The reset value of this field is 0xA.

## B.2.19 TPIU\_PIDR5, Peripheral Identification Register 5

The TPIU\_PIDR5 register is reserved.

## Usage constraints

Access is only allowed from privileged code. Unprivileged access results in a BusFault being raised.

## Configurations

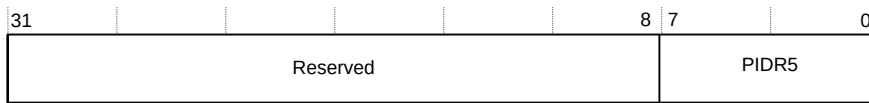
This register is always implemented when the TPIU is included.

## Attributes

This is a 32-bit register. See [TPIU register summary](#) for more information.

The following figure shows the TPIU\_PIDR5 bit assignments.

### Figure B-20: TPIU\_PIDR5 bit assignments



The following table describes the TPIU\_PIDR5 bit assignments.

### Table B-21: TPIU\_PIDR5 bit assignments

Field	Name	Type	Description
[31:8]	Reserved	-	RES0.
[7:0]	PIDR5	RO	RES0.

### B.2.20 TPIU\_PIDR6, Peripheral Identification Register 6

The TPIU\_PIDR6 register is reserved.

## Usage constraints

Access is only allowed from privileged code. Unprivileged access results in a BusFault being raised.

## Configurations

This register is always implemented when the TPIU is included.

## Attributes

This is a 32-bit register. See [TPIU register summary](#) for more information.

The following figure shows the TPIU\_PIDR6 bit assignments.

**Figure B-21: TPIU\_PIDR6 bit assignments**



The following table describes the TPIU PIDR6 bit assignments.

### Table B-22: TPIU\_PIDR6 bit assignments

Field	Name	Type	Description
[31:8]	Reserved	-	RES0.

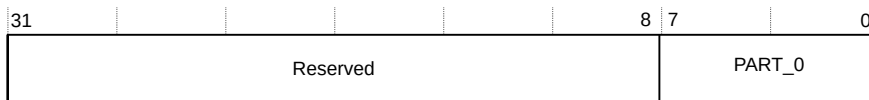


## Attributes

This is a 32-bit register. See [TPIU register summary](#) for more information.

The following figure shows the TPIU\_PIDR0 bit assignments.

### Figure B-23: TPIU\_PIDR0 bit assignments



The following table describes the TPIU\_PIDR0 bit assignments.

### Table B-24: TPIU\_PIDR0 bit assignments

Field	Name	Type	Description
[31:8]	Reserved	-	<b>RES0.</b>
[7:0]	PART_0	RO	<p>This field indicates the part number. When taken together with TPIU_PIDR1.PART_1, it indicates the component. The part number is selected by the designer of the component.</p> <p>The reset value of this field is 0x24.</p>

### B.2.23 TPIU\_PIDR1, Peripheral Identification Register 1

The TPIU\_PIDR1 register indicates the TPIU component JEP106 continuation code and part number.

## Usage constraints

Access is only allowed from privileged code. Unprivileged access results in a BusFault being raised.

## Configurations

This register is always implemented when the TPIU is included.

## Attributes

This is a 32-bit register. See [TPIU register summary](#) for more information.

The following figure shows the TPIU\_PIDR1 bit assignments.



The following table describes the TPIU\_PIDR2 bit assignments.

**Table B-26: TPIU\_PIDR2 bit assignments**

Field	Name	Type	Description
[31:8]	Reserved	-	<b>RES0.</b>
[7:4]	REVISION	RO	This field indicates the revision number of the TPIU component. It is an incremental value starting at 0x0 for the first design. The reset value is 0x0.
[3]	JEDEC	RO	This field is always 1, indicating that a JEDEC assigned value is used.
[2:0]	DES_1	RO	This field is the JEP106 identification code, bits[6:4]. Together, with TPIU_PIDR4.DES_2 and TPIU_PIDR1.DES_0, they indicate the designer of the component and not the implementer, except where the two are the same. The reset value is 0b111.

## B.2.25 TPIU\_PIDR3, Peripheral Identification Register 3

The TPIU\_PIDR3 register indicates minor errata fixes of the TPIU component and if you have modified the behavior of the component.

### Usage constraints

Access is only allowed from privileged code. Unprivileged access results in a BusFault being raised.

### Configurations

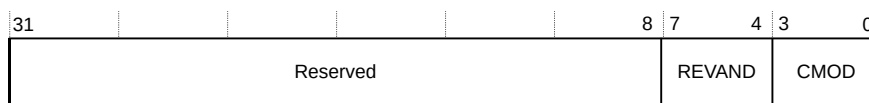
This register is always implemented when the TPIU is included.

### Attributes

This is a 32-bit register. See [TPIU register summary](#) for more information.

The following figure shows the TPIU\_PIDR3 bit assignments.

**Figure B-26: TPIU\_PIDR3 bit assignments**



The following table describes the TPIU\_PIDR3 bit assignments.

**Table B-27: TPIU\_PIDR3 bit assignments**

Field	Name	Type	Description
[31:8]	Reserved	-	<b>RES0.</b>
[7:4]	REVAND	RO	This field indicates minor errata fixes specific to this design, for example metal fixes after implementation. In most cases this field is 0x0.

Field	Name	Type	Description
[3:0]	CMOD	RO	Customer modified. Where the component is reusable IP, this value indicates whether you have modified the behavior of the component. In most cases, this field is 0x0.

## B.2.26 TPIU\_CIDR0, Component Identification Register 0

The TPIU\_CIDR0 register indicates the preamble.

### Usage constraints

Access is only allowed from privileged code. Unprivileged access results in a BusFault being raised.

### Configurations

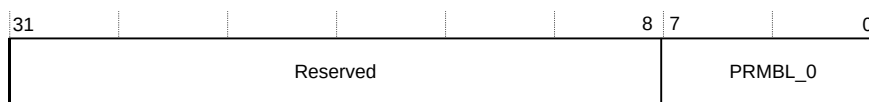
This register is always implemented when the TPIU is included.

### Attributes

This is a 32-bit register. See [TPIU register summary](#) for more information.

The following figure shows the TPIU\_CIDR0 bit assignments.

**Figure B-27: TPIU\_CIDR0 bit assignments**



The following table describes the TPIU\_CIDR0 bit assignments.

**Table B-28: TPIU\_CIDR0 bit assignments**

Field	Name	Type	Description
[31:8]	Reserved	-	RES0.
[7:0]	PRMBL_0	RO	Preamble. This field returns 0x0D.

## B.2.27 TPIU\_CIDR1, Component Identification Register 1

The TPIU\_CIDR1 register indicates the component class and preamble.

### Usage constraints

Access is only allowed from privileged code. Unprivileged access results in a BusFault being raised.

### Configurations

This register is always implemented when the TPIU is included.





The following table describes the TPIU\_CIDR2 bit assignments.

**Table B-30: TPIU\_CIDR2 bit assignments**

Field	Name	Type	Description
[31:8]	Reserved	-	RES0.
[7:0]	PRMBL_2	RO	Preamble. This field returns 0x05.

## B.2.29 TPIU\_CIDR3, Component Identification Register 3

The TPIU\_CIDR3 register indicates the preamble.

### Usage constraints

Access is only allowed from privileged code. Unprivileged access results in a BusFault being raised.

### Configurations

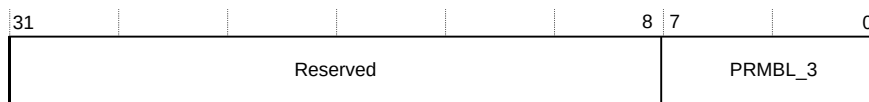
This register is always implemented when the TPIU is included.

### Attributes

This is a 32-bit register. See [TPIU register summary](#) for more information.

The following figure shows the TPIU\_CIDR3 bit assignments.

**Figure B-30: TPIU\_CIDR3 bit assignments**



The following table describes the TPIU\_CIDR3 bit assignments.

**Table B-31: TPIU\_CIDR3 bit assignments**

Field	Name	Type	Description
[31:8]	Reserved	-	RES0.
[7:0]	PRMBL_3	RO	Preamble. This field returns 0xB1.

# Appendix C Signal Descriptions

This appendix describes the Cortex®-M52 processor signals.

## C.1 Clock and clock enable signals

The following table shows the Cortex®-M52 processor clock and clock enable signals.

**Table C-1: Clock and clock enable signals**

Signal name	Direction	Description
CLKIN	Input	Primary processor clock. This is gated internally for functional units when required depending on the operating mode of the processor.
DBGCLK	Input	Clock driving the majority of the debug and trace logic in the processor.
SSTCLKEN	Input	Synchronous enable that is used with CLKIN to derive the secure system SysTick clock.
NSSTCLKEN	Input	Synchronous enable that is used with CLKIN to derive the Non-secure system SysTick clock.
IWICCLK	Input	This signal is the IWIC clock.

## C.2 Reset signals

The following table shows the Cortex®-M52 processor reset signals.

**Table C-2: Reset signals**

Signal name	Direction	Description
nPORESET	Input	<p>Cold reset.</p> <p>If <i>Dual-Core Lock-step</i> (DCLS) is not configured in the processor, the nPORESET signal is treated as an asynchronous input. Reset assertion is fully asynchronous and does not require an active clock. Reset de-assertion is synchronized inside the processor.</p> <p>If DCLS is configured in the processor, this signal must be asserted and deasserted together with nPORESETDCLS. If CLKIN is active when nPORESET is asserted or deasserted, then the signal must be constrained such that nPORESET is stable on the rising edge of the clock.</p> <p>For more information on nPORESETDCLS, see <a href="#">DCLS operation signals</a></p>

Signal name	Direction	Description
nSYSRESET	Input	<p>System reset.</p> <p>This signal resets non-debug logic and all memory interfaces except for the <i>Debug-AHB</i> (D-AHB) and External Private Peripheral Bus (EPPB) interfaces.</p> <p>If DCLS is not configured in the processor, the nSYSRESET signal is treated as an asynchronous input. Reset assertion is fully asynchronous and does not require an active clock. Reset de-assertion is synchronized inside the processor.</p> <p>If DCLS is configured in the processor, this signal must be asserted and deasserted together with nSYSRESETDCLS. If CLKIN is active when nSYSRESET is asserted or deasserted, then the signal must be constrained such that nSYSRESET is stable on the rising edge of the clock.</p> <p>For more information on nSYSRESETDCLS, see <a href="#">DCLS operation signals</a></p>
nDBGRESET	Input	<p>Debug reset that resets all logic in the debug power domain (PDDEBUG). This reset must be asserted at Cold reset along with nPORESET and when PDDEBUG is powered down.</p>
nIWICRESET	Input	<p>This is an active-LOW IWIC reset signal. This signal is internally synchronized to IWICCLK.</p> <p>If Dual-Core Lockstep (DCLS) is not configured in the processor, the nIWICRESET signal is treated as an asynchronous input. Reset assertion is fully asynchronous and does not require an active clock. Reset de-assertion is synchronized inside the processor.</p> <p>If DCLS is configured in the processor, by setting the Verilog parameter LOCKSTEP, this signal must be asserted and deasserted together with nIWICRESETDCLS. If IWICCLK is active when nIWICRESET is asserted or deasserted, then the signal must be constrained such that nIWICRESET is stable on the rising edge of the clock.</p> <p>For more information on nIWICRESETDCLS, see <a href="#">DCLS operation signals</a></p>
nMBISTRESET	Input	<p>Production MBIST reset.</p>

## C.3 Static configuration signals

The following table shows the Cortex®-M52 processor static configuration signals.

The configuration signals in the following table can only be changed at Cold reset with nPORESET asserted. They are intended to be static configuration signals that are fixed for a given integration of the processor.

**Table C-3: Static configuration signals**

Signal name	Direction	Description
CFGITCMSZ[3:0]	Input	<p>Size of the <i>Instruction Tightly Coupled Memory</i> (ITCM) region encoded as:</p> <p><b>CFGITCMSZ = 0b0000</b> ITCM is not implemented.  <b>CFGITCMSZ &gt; 0b0010</b> <math>2^{\text{CFGITCMSZ}-1}\text{KB}</math></p> <ul style="list-style-type: none"> <li>The minimum size of <i>Tightly Coupled Memory</i> (TCM) is 4KB and the maximum size is 16MB. Setting CFGITCMSZ to 0b0001 or 0b0010 results in <b>UNPREDICTABLE</b> behavior.</li> <li>The CFGITCMSZ input signal sets the ITCM size. The ITGUMAXBLKS parameter constraints the maximum ITCM size that can be used. Therefore, the ITGUMAXBLKS must be set to be large enough to accommodate the anticipated ITCM size that might be used in the system.</li> </ul>

Signal name	Direction	Description
CFGDTCMSZ[3:0]	Input	<p>Size of the <i>Data Tightly Coupled Memory</i> (DTCM) region encoded as:</p> <div><div><p><b>CFGDTCMSZ = 0b0000</b> <b>CFGDTCMSZ &gt; 0b0010</b></p></div><div><p>DTCM is not implemented. <math>2^{\text{CFGDTCMSZ}-1}\text{KB}</math></p></div></div> <ul style="list-style-type: none"><li>The CFGDTCMSZ input signal sets the DTCM size. The DTGUMAXBLS parameter constraints the maximum DTCM size that can be used. Therefore, the DTGUMAXBLS must be set to be large enough to accommodate the anticipated DTCM size that might be used in the system.</li><li>The minimum size of the TCM is 4KB and the maximum size is 16MB. Setting CFGDTCMSZ to 0b0001 or 0b0010 results in <b>UNPREDICTABLE</b> behavior.</li></ul>
CFGPAHBSZ[2:0]	Input	<p>Size of the <i>Peripheral AHB</i> (P-AHB) peripheral port memory region.</p> <div><div><p><b>0bb000</b> <b>0bb001</b> <b>0bb010</b> <b>0bb011</b> <b>0bb100</b></p></div><div><p>P-AHB disabled. 64MB 128MB 256MB 512MB</p></div></div> <p>Setting CFGPAHBSZ to any other value results in <b>UNPREDICTABLE</b> behavior.</p>
CFGMEMALIAS[2:0]	Input	<p>Memory address alias bit for the ITCM, DTCM, and P-AHB regions. The address bit used for the memory alias is determined by:</p> <div><div><p><b>0b001</b> <b>0b010</b> <b>0b100</b> <b>0b000</b></p></div><div><p>Alias bit = 26 Alias bit = 27 Alias bit = 28 No alias. TCM security gating is disabled.</p></div></div> <p>Default invalid. Setting CFGMEMALIAS to an invalid value will result in <b>UNPREDICTABLE</b> behavior.</p>
CFGCPUINST[1:0]	Input	<p>The current Cortex®-M52 instantiate ID. This signal is used in a system that 1-4 Cortex®-M52 copies are instantiated. These bits are used as TCM base address [25:24].</p>
CFGFPMVE[2:0]]	Input	<p>Specifies the floating point and <i>M-profile Vector Extension</i> (MVE) features of the processor.</p> <div><div><p><b>0</b> <b>1</b> <b>2</b> <b>3</b> <b>4</b> <b>5</b></p></div><div><p>No floating point. No MVE. Scalar half and single precision floating point. No MVE. Scalar half, single and double precision floating point. No MVE. No floating point. Integer subset of MVE. Scalar half and single precision floating point. Integer subset of MVE. Scalar half, single and double precision floating point. Integer and half and single precision floating point MVE.</p></div></div> <p><b>Note:</b> All other values are invalid.</p>

Signal name	Direction	Description
CFGBIGEND	Input	This signal is used to select the data endian format.  <b>0</b> Little-endian (LE). <b>1</b> Byte-invariant big-endian (BE8).
MPUNSDISABLE	Input	If Non-secure memory regions are configured for the <i>Memory Protection Unit</i> (MPU), disables support for the Non-secure MPU region.
MPUSDISABLE	Input	If Secure regions are configured for the MPU, disables support for the Secure MPU region.
SAUDISABLE	Input	If the <i>Security Attribution Unit</i> (SAU) is configured, disables support.
CFGNOCDECP[7:0]	Input	Disables support of CDE onto coprocessor instructions.  If the Verilog parameter <code>CDEMAPPEDONCPn</code> is set to 1, setting <code>CFGNOCDECP[n]</code> to 1 will force coprocessor instruction behavior for CPn.
CFGSTCALIB[25:0]	Input	Secure SysTick calibration configuration: <ul style="list-style-type: none"> <li>CFGSTCALIB[23:0]: TENMS</li> <li>CFGSTCALIB[24]: SKEW</li> <li>CFGSTCALIB[25]: NOREF</li> </ul>
CFGNSTCALIB[25:0]	Input	Non-secure SysTick calibration configuration: <ul style="list-style-type: none"> <li>CFGNSTCALIB[23:0]: TENMS</li> <li>CFGNSTCALIB[24]: SKEW</li> <li>CFGNSTCALIB[25]: NOREF</li> </ul>

## C.4 Reset configuration signals

The following table shows the Cortex®-M52 processor reset configuration signals. These signals are sampled at deassertion of Warm reset or Cold reset, and their values can change out of reset. The reset configuration signals can be used more dynamically than the static configuration signals.

**Table C-4: Reset configuration signals**

Signal name	Direction	Description
INITSVTOR[31:7]	Input	This signal indicates the Secure vector table offset address out of reset, <code>VTOR_S.TBLOFF[31:7]</code> . For more information on <code>VTOR_S</code> , see the <i>Arm®v8-M Architecture Reference Manual</i> .  When <code>SECEXT=0</code> , <code>VTOR_S</code> and associated signals still exist but are not used, and only <code>VTOR_NS</code> and its associated signals are used.
INITNSVTOR[31:7]	Input	This signal indicates the Non-secure vector table offset address out of reset, <code>VTOR_NS.TBLOFF[31:7]</code> . For more information on <code>VTOR_NS</code> , see the <i>Arm®v8-M Architecture Reference Manual</i> .
INITTCMEN[1:0]	Input	<i>Tightly Coupled Memory</i> (TCM) enable initialization out of reset:  <b>Bit[0] is HIGH:</b> <i>Instruction Tightly Coupled Memory</i> (ITCM) is enabled. <b>Bit[1] is HIGH:</b> <i>Data Tightly Coupled Memory</i> (DTCM) is enabled.  This signal controls the reset value of <code>ITCMCR.EN</code> and <code>DTCMCR.EN</code> bits. For more information on <code>ITCMCR</code> and <code>DTCMCR</code> , see the <i>Arm® Cortex®-M52 Processor Technical Reference Manual</i> .

Signal name	Direction	Description
INITPAHBEN	Input	<p>P-AHB enable initialization out of reset:</p> <p><b>HIGH</b> P-AHB is enabled. <b>LOW</b> P-AHB is disabled.</p> <p>For more information on PAHBSCR, see the <i>Arm® Cortex®-M52 Processor Technical Reference Manual</i>.</p>
INITECCEN	Input	<p>TCM and L1 cache <i>Error Correcting Code</i> (ECC) enable out of Cold reset.</p> <p><b>HIGH</b> ECC is enabled. <b>LOW</b> ECC is disabled.</p> <p>If ECC is not configured in the processor, this signal has no effect on the processor.</p> <p>ECC must not be enabled dynamically when the processor is in the Memory retention mode (MEM_RET) power mode. This is because the L1 cache is not automatically invalidated with the Memory retention mode power mode is switched on. This results in inconsistent ECC information that is relative to the data that is retained in the cache. This results in an ECC error.</p>

## C.5 Cache initialization signal

The data and instruction caches can be automatically initialized when enabled at reset or if the PDRAMS power domain is enabled during runtime. This functionality can be disabled if required using the INITL1RSTDIS signal. The following table describes the INITL1RSTDIS signal.

**Table C-5: Cache initialization signal**

Signal name	Direction	Description
INITL1RSTDIS	Input	<p>Disable L1 cache invalidation out of reset.</p> <p><b>HIGH</b> Disable automatic invalidation of the L1 cache. <b>LOW</b> Enable automatic invalidation of the L1 cache that occurs in the following cases:</p> <ul style="list-style-type: none"> <li>The P-Channel is used to turn on the PDCORE domain. Power mode transitions from OFF to ON. Invalidation does not occur on transitions from OFF to MEM_RET or MEM_RET to ON.</li> <li>nSYSRESET is asserted when the PDRAMS are powered on, that is, when the processor is in either of the following: <ul style="list-style-type: none"> <li>The power modes ON (cache). Arm® does not recommend that you assert nSYSRESET when the processor is ON (Cache) because this can cause a system error.</li> <li>The WARM_RST power mode with PDRAMS on.</li> </ul> </li> <li>The P-Channel is used to move the power mode from ON (no cache) to ON (cache).</li> </ul> <p><b>Note:</b></p> <ul style="list-style-type: none"> <li>If the P-Channel is used to control the processor power mode selection, then this signal must be tied LOW unless valid cache RAM content is required to be preserved after WARM_RST.</li> <li>If INITECCEN is HIGH, this signal must be LOW on reset unless the content of the instruction and data cache tag RAMs is guaranteed to be valid.</li> </ul> <p>For more information on the P-Channel and power modes, see <a href="#">Power management</a>.</p>

## C.6 Instruction execution control signals

The following table shows the instruction execution control signals that must be connected in your *System on Chip* (SoC) design.

**Table C-6: Instruction execution control signals**

Signal name	Direction	Description														
CPUWAIT	Input	Stall the core out of reset.														
CURRNS	Output	<p>Current Security state of the Cortex®-M52 processor:</p> <p><b>HIGH</b> Processor is in Non-secure state. <b>LOW</b> Processor is in Secure state.</p> <p>If the Cortex®-M52 processor is not configured for Security Extension support, this signal is always asserted.</p>														
CURRPC[31:1]	Output	<p>This signal is the address of the current instruction the processor is executing.</p> <p><b>Note:</b> CURRNS indicates the Security state of the executing instruction.</p>														
FAULTSTAT[42:0]	Output	<p>This signal is asserted when the processor detects a fault while an exception is in progress. The signal encodes all the following Fault Status Registers:</p> <table><tr><td><b>FAULTSTAT[42:35]</b></td><td>SFSR[7:0]</td></tr><tr><td><b>FAULTSTAT[34]</b></td><td>HFSR.DEBUGEVT</td></tr><tr><td><b>FAULTSTAT[33]</b></td><td>HFSR.FORCED</td></tr><tr><td><b>FAULTSTAT[32]</b></td><td>HFSR.VECTTBL</td></tr><tr><td><b>FAULTSTAT[31:16]</b></td><td>UFSR</td></tr><tr><td><b>FAULTSTAT[15:8]</b></td><td>BFSR</td></tr><tr><td><b>FAULTSTAT[7:0]</b></td><td>MMFSR</td></tr></table> <p><b>Note:</b> This signal is not fully synchronous with the detection of the fault inside the processor.</p>	<b>FAULTSTAT[42:35]</b>	SFSR[7:0]	<b>FAULTSTAT[34]</b>	HFSR.DEBUGEVT	<b>FAULTSTAT[33]</b>	HFSR.FORCED	<b>FAULTSTAT[32]</b>	HFSR.VECTTBL	<b>FAULTSTAT[31:16]</b>	UFSR	<b>FAULTSTAT[15:8]</b>	BFSR	<b>FAULTSTAT[7:0]</b>	MMFSR
<b>FAULTSTAT[42:35]</b>	SFSR[7:0]															
<b>FAULTSTAT[34]</b>	HFSR.DEBUGEVT															
<b>FAULTSTAT[33]</b>	HFSR.FORCED															
<b>FAULTSTAT[32]</b>	HFSR.VECTTBL															
<b>FAULTSTAT[31:16]</b>	UFSR															
<b>FAULTSTAT[15:8]</b>	BFSR															
<b>FAULTSTAT[7:0]</b>	MMFSR															



## C.7 Instruction Tightly Coupled Memory interface signals

The following table shows the Cortex®-M52 processor *Instruction Tightly Coupled Memory* (ITCM) interface signals. If you do not use the ITCM in your SoC, you must tie all the ITCM interface input signals to LOW.

**Table C-7: ITCM interface signals**

Signal name	Direction	Phase	Description
ITCMADDR[23:2]	Output	Address	Transfer address for reads and writes.  All ITCM accesses are 32-bit aligned. If necessary, the processor selects read data based on the full address.
ITCMCS	Output	Address	RAM chip select.
ITCMPRIV	Output	Address	Privilege level of access:  <b>0</b> User access. <b>1</b> Privileged access.
ITCMWR	Output	Address	RAM write enable:  <b>0</b> Read access request. <b>1</b> Write-Access request.  Valid when ITCMCS is HIGH.
ITCMBYTEWR[4:0]	Output	Address	Byte write strobes.  <b>n&lt;4</b> Bit[n] is to indicate data bits [8n+7:8n]. <b>n=4</b> Bit[n] is to indicate that <i>Error Correcting Code</i> (ECC) information is written in ITCMWDATA[38:32].  This signal is valid when ITCMCS is HIGH.  <b>Note:</b> If ITCMWR is 0b0, ITCMBYTEWR = 0b0.
ITCMWDATA[38:0]	Output	Address	<b>ITCMWDATA[31:0]</b> Write data (32-bits). <b>ITCMWDATA[38:32]</b> ECC information (7-bits).  ITCMBYTEWR defines validity of this signal on a byte-wise basis, otherwise, memory ignores this signal.  If ECC is not configured, ITCMWDATA[38:32] can be left unconnected.

Signal name	Direction	Phase	Description
ITCMMASTER[3:0]	Output	Address	<p>Encodes the requestor of the current access:</p> <p><b>0b0000</b> Instruction fetch.  <b>0b0001</b> Data that is read from software on the processor.  <b>0b0010</b> Vector fetch on exception entry.  <b>0b0011</b> Read from <i>AHB TCM Access</i> (TCM-AHB).  <b>0b0100</b> Debugger read.  <b>0b0101</b> <i>Memory Built-In Self Test</i> (MBIST) access.  <b>0b1001</b> Data write from software on the processor, including <i>Read Modify Write</i> (RMW) read access.  <b>0b1011</b> Debugger write.  <b>0b1100</b> ECC correction.  <b>0b1101</b> Stack pointer vector fetch, indicating that the TCM access is associated with reading the initial stack pointer from the reset vector.  <b>0b1110</b> Write from TCM-AHB, including RMW read access.</p> <p>Can be used to monitor debug requests or used to change the behavior of TCM accesses for debug.</p>
ITCMRDATA[38:0]	Input	Response	<p><b>ITCMRDATA[31:0]</b> Read data (32-bits).  <b>ITCMRDATA[38:32]</b> ECC information (7-bits).</p> <p>All data bytes are valid on the last cycle of a read response phase. The processor ignores this signal on all other cycles.</p>
ITCMWAIT	Input	Response	<p>Wait signal to extend the current response phase:</p> <p><b>0</b> Complete phase.  <b>1</b> Extend phase.</p>
ITCMERR	Input	Response	<p>Error indication for the current transaction, valid on the last cycle of the response phase.</p> <p><b>0</b> No error.  <b>1</b> Error.</p>

## C.8 Data Tightly Coupled Memory interface signals

The following table shows the Cortex®-M52 processor *Data Tightly Coupled Memory* (DTCM) interface signals. If you are not using DTCM in your SoC, you must tie all the DTCM interface input signals to LOW.

**Table C-8: DTCM interface signals**

Signal name	Direction	Phase	Description
D*TCMADDR[23:3]	Output	Address	<p>Transfer address for both reads and writes.</p> <p>All DTCM accesses are 32-bit aligned. The processor selects read data as required based on the full address.</p>
D*TCMCS	Output	Address	RAM chip select.

Signal name	Direction	Phase	Description
D*TCMPRIV	Output	Address	<p>Privilege level of access:</p> <p><b>0</b> User access. <b>1</b> Privileged access.</p>
D*TCMWR	Output	Address	<p>RAM write enable:</p> <p><b>0</b> Read access request. <b>1</b> Write access request.</p> <p>Valid when D*TCMCS is HIGH.</p>
D*TCMBYTEWR[4:0]	Output	Address	<p>Byte write strobes.</p> <p><b>n&lt;4</b> Bit[n] is to indicate data bits [8n+7:8n]. <b>n=4</b> Bit[n] is to indicate that <i>Error Correcting Code</i> (ECC) information is written in D*TCMWDATA[38:32].</p> <p>This signal is valid when D*TCMCS is HIGH.</p> <p><b>Note:</b> If D*TCMWR is 0, D*TCMBYTEWR is 0x0.</p>
D*TCMWDATA[38:0]	Output	Address	<ul style="list-style-type: none"> <li>D*TCMWDATA[31:0]: Write data (32-bits).</li> <li>D*TCMWDATA[38:32]: <i>Error Correcting Code</i> (ECC) information (7-bits).</li> </ul> <p>D*TCMBYTEWR defines validity of this signal on a byte-wise basis, otherwise memory ignores this signal. If ECC is not configured, D*TCMWDATA[38:32] can be left unconnected.</p>
D*TCMASTER[3:0]	Output	Address	<p>Encodes the requestor of the current access:</p> <p><b>0b0000</b> Instruction fetch. <b>0b0001</b> Data that is read from software on the processor. <b>0b0010</b> Vector fetch on exception entry. <b>0b0011</b> Read from <i>AHB TCM Access</i> (TCM-AHB). <b>0b0100</b> Debugger read. <b>0b0101</b> <i>Memory Built-In Self Test</i> (MBIST) access. <b>0b1001</b> Data write from software on the processor, including <i>Read Modify Write</i> (RMW) read access. <b>0b1011</b> Debugger write. <b>0b1100</b> ECC correction. <b>0b1101</b> Stack pointer vector fetch, indicating that the TCM access is associated with reading the initial stack pointer from the reset vector. <b>0b1110</b> Write from TCM-AHB including RMW read access.</p> <p>Can be used to monitor debug requests or used to change the behavior of TCM accesses for debug.</p>
D*TCMRDATA[38:0]	Input	Response	<ul style="list-style-type: none"> <li>D*TCMRDATA[31:0]: Read data (32-bits).</li> <li>D*TCMRDATA[38:32]: ECC information (7-bits).</li> </ul> <p>All data bytes are valid on the last cycle of a read response phase. The processor ignores this signal on all other cycles.</p>

Signal name	Direction	Phase	Description
D*TCMWAIT	Input	Response	Wait signal to extend the current data phase:  <div> <div>0</div> <div>1</div> </div> <div> <div>Complete phase.</div> <div>Extend phase.</div> </div>
D*TCMERR	Input	Response	Error indication for the current transaction, valid on the last cycle of the response phase.  <div> <div>0</div> <div>1</div> </div> <div> <div>No error.</div> <div>Error.</div> </div>

## C.9 M-AXI interface signals

The AXI Main (M-AXI) interface implements the standard set of AMBA® 5 AXI read and write channel signals.

The following table shows the M-AXI master interface signals. For more information on the AMBA AXI signals, see the *AMBA® AXI and ACE Protocol Specification*.

**Table C-9: M-AXI interface signals**

Signal name	Direction	Description
ACLKEN	Input	Clock enable for the AXI port. Supports semi-synchronous operation of the interface relative to the processor clock.  <b>Note:</b> ACLKEN can be used to clock all other M-AXI signals at an integer division of the processor clock. This includes support for timing the interface at n:1 for all other signals.
AWAKEUP	Output	Indicates that the master starts a transaction and sends it to the interconnect.
AWVALID	Output	Write address valid signal.
AWADDR[31:0]	Output	Write address signal.
AWBURST[1:0]	Output	Write burst type signal.
AWLEN[2:0]	Output	Write burst length signal.
AWSIZE[1:0]	Output	Write burst size signal.
AWLOCK	Output	Write lock type signal.
AWPROT[2:0]	Output	Write protection type signal.
AWREADY	Input	Write address ready signal.
AWID[1:0]	Output	Write request ID signal.  <div> <div>0b00</div> <div>0b01</div> <div>0b10</div> <div>0b11</div> </div> <div> <div>Writes to Normal Non-cacheable memory and all store-exclusive transactions.</div> <div>Writes to cacheable memory.</div> <div>Writes to Device memory.</div> <div>Cache line evictions.</div> </div>
AWCACHE[3:0]	Output	Outer Cacheability attributes. For more information on the encoding of this signal, see the <i>AMBA® AXI and ACE Protocol Specification</i> .

Signal name	Direction	Description
AWINNER[3:0]	Output	Inner Cacheability attributes. The encoding is identical to AWCACHE[3:0]. For more information on the encoding of AWCACHE[3:0] signal, see the <i>AMBA® AXI and ACE Protocol Specification</i> .
AWDOMAIN[1:0]	Output	<p>Inner and outer Shareability attributes as defined in the active memory map.</p> <p> <b>0b00</b> Non-shareable  <b>0b01</b> Reserved  <b>0b10</b> Inner Shareable and Outer Shareable  <b>0b11</b> System </p> <p>For more information on the encoding of this signal, see the <i>AMBA® AXI and ACE Protocol Specification</i>.</p>
AWSPARSE	Output	Transaction might use sparse writes strobes. This signal indicates a write burst which might contain a beat which includes sparse data. That is, a beat which cannot be directly translated into an AHB transaction. If the signal is LOW, then the burst is guaranteed to be made up of contiguous and appropriately aligned data relative to data size.
AWMASTER	Output	<p>Initiator of access.</p> <p> <b>0</b> Processor access.  <b>1</b> Debugger access. </p>
ARVALID	Output	Read address valid signal.
ARADDR[31:0]	Output	Read address signal.
ARBURST[1:0]	Output	Read burst type signal.
ARLEN[7:0]	Output	Read address burst length signal.
ARSIZE[1:0]	Output	Read burst size signal.
ARLOCK	Output	Read lock type signal.
ARPROT[2:0]	Output	Read protection type signal.
ARREADY	Input	Read address ready signal.
ARID[2:0]	Output	<p>Read request ID signal.</p> <p> <b>0b000</b> All accesses to Non-cacheable and Device memory regions (including bursts).  <b>0b001</b> Unified cache linefills for load.  <b>0b010</b> Data cache linefills from linefill buffer.  <b>0b100</b> Instruction fetch or instruction linefill and vector fetch on exception entry. </p>
ARCACHE[3:0]	Output	Outer Cacheability attributes. For more information on the encoding of this signal, see the <i>AMBA® AXI and ACE Protocol Specification</i> .
ARINNER[3:0]	Output	Inner Cacheability attributes. The encoding is identical to ARCACHE[3:0]. For more information on the encoding of ARCACHE[3:0] signal, see the <i>AMBA® AXI and ACE Protocol Specification</i> .
ARDOMAIN[1:0]	Output	<p>Inner and Outer Shareability attributes as defined in the active memory map.</p> <p> <b>0b00</b> Non-shareable  <b>0b01</b> Reserved  <b>0b10</b> Inner Shareable and Outer Shareable  <b>0b11</b> System </p> <p>For more information on the encoding of this signal, see the <i>AMBA® AXI and ACE Protocol Specification</i>.</p>
ARMASTER	Output	<p>Initiator of access.</p> <p> <b>0</b> Processor access.  <b>1</b> Debugger access. </p>

Signal name	Direction	Description
WID[1:0]	Output	Write data ID signal. Used to connect to AXI3 interconnect or slaves.  Can be ignored for AXI4 or AXI5 interconnect or slaves.  <b>0b00</b> Writes to Normal Non-cacheable memory and all store-exclusive transactions. <b>0b01</b> Writes to cacheable memory. <b>0b10</b> Writes to Device memory. <b>0b11</b> Cache line evictions.
WVALID	Output	Write data valid signal.
WLAST	Output	Indicates last transfer in a write burst.
WSTRB[3:0]	Output	Write byte lane strobes.
WDATA[31:0]	Output	Write data signal.
WPOISON	Output	Indicates that a set of data bytes has been corrupted.
WREADY	Input	Write data ready signal.
RVALID	Input	Read data valid signal.
RID[2:0]	Input	Read data ID.  <b>0b000</b> All accesses to Non-cacheable and Device memory regions (including bursts). <b>0b001</b> Unified cache linefills for load. <b>0b010</b> Data cache linefills from linefill buffer. <b>0b100</b> Instruction fetch or instruction linefill.
RLAST	Input	Indicates last transfer in read data.
RDATA[31:0]	Input	Read data.
RRESP[1:0]	Input	Read data response.
RPOISON	Input	Indicates that a set of data bytes has been corrupted.
RREADY	Output	Read data ready signal.
BVALID	Input	Write response valid signal.
BID[1:0]	Input	Write response ID signal.  <b>0b00</b> Writes to Normal Non-cacheable memory and all store-exclusive transactions. <b>0b01</b> Writes to cacheable memory. <b>0b10</b> Writes to Device memory. <b>0b11</b> Cache line evictions.
BRESP[1:0]	Input	Write response signal.
BREADY	Output	Write response ready signal.

## C.9.1 M-AXI interface protection signals

The following table shows the M-AXI interface protection signals.

**Table C-10: M-AXI interface protection signals**

Signal name	Direction	Description
ACLKENCHK	Input	Odd parity of ACLKEN.
AWAKEUPCHK	Output	Odd parity of AWAKEUP
ARVALIDCHK	Output	Odd parity of ARVALID.

Signal name	Direction	Description
ARREADYCHK	Input	Odd parity of ARREADY.
ARADDRCHK[3:0]	Output	Odd parity of ARADDR[31:0] at 8-bit granularity.
ARIDCHK	Output	Odd parity of ARID[2:0].
ARLENCHK	Output	Odd parity of ARLEN[3:0].
ARUSERCHK	Output	Odd parity of (ARINNER[3:0], ARMASTER).
ARCTLCHK0	Output	Odd parity of (ARSIZE[2:0], ARBURST[1:0], ARLOCK, ARPROT[2:0]).
ARCTLCHK1	Output	Odd parity of ARCACHE[3:0].
ARCTLCHK2	Output	Odd parity of ARDOMAIN[1:0].
AWVALIDCHK	Output	Odd parity of AWVALID.
AWREADYCHK	Input	Odd parity of AWREADY.
AWADDRCHK[3:0]	Output	Odd parity of AWADDR[31:0] at 8-bit granularity.
AWIDCHK	Output	Odd parity of AWID[1:0].
AWLENCHK	Output	Odd parity of AWLEN[3:0].
AWUSERCHK	Output	Odd parity of (AWSPARSE, AWINNER[3:0], AWMMASTER).
AWCTLCHK0	Output	Odd parity of (AWSIZE[2:0], AWBURST[1:0], ARLOCK, ARPROT[2:0]).
AWCTLCHK1	Output	Odd parity of (AWCACHE[3:0], AWPROT[2:0], AWLOCK).
AWCTLCHK2	Output	Odd parity of AWDOMAIN[1:0].
RDATACHK[3:0]	Input	This signal can be used to detect, and potentially correct data bytes that might be corrupted.
RVALIDCHK	Input	Odd parity of RVALID.
RREADYCHK	Output	Odd parity of RREADY.
RIDCHK	Input	Odd parity of RID[2:0].
RLASTCHK	Input	Odd parity of RLAST.
RRESPCHK	Input	Odd parity of RRESP[1:0].
RPOISONCHK	Input	Odd parity of RPOISON.
WDATACHK[3:0]	Output	This signal can be used to detect, and potentially correct data bytes that might be corrupted.
WVALIDCHK	Output	Odd parity of WVALID.
WREADYCHK	Input	Odd parity of WREADY.
WSTRBCHK	Output	Odd parity ofWSTRB[7:0].
WIDCHK	Output	Odd parity of WID[1:0].
WLASTCHK	Output	Odd parity of WLAST.
WPOISONCHK	Output	Odd parity of WPOISON.
BVALIDCHK	Input	Odd parity of BVALID.
BREADYCHK	Output	Odd parity of BREADY.
BIDCHK	Input	Odd parity of BID[2:0].
BRESPCHK	Input	Odd parity of BRESP[1:0].

## C.10 CODE-AHB interface signals

Cortex®-M52 can be configured to either use AXI interface or AHB interface. If AHB interface is chosen, then the CODE-AHB interface and SYS-AHB interface are used instead of the AXI interface.

The following table shows the CODE-AHB interface signals.

**Table C-11: CODE-AHB interface signals**

Signal name	Direction	Timing	Description
HTRANSC[1:0]	Out	50%	Transfer type
HBURSTC[2:0]	Out	60%	Transfer burst length
HADDR[31:0]	Out	60%	Transfer address
HWRITEC	Out	60%	Write transfer
HSIZEC[2:0]	Out	60%	Transfer size
HWDATAC[31:0]	Out	60%	Write data
HPROTC[6:0]	Out	60%	Protection and outer memory attributes
HNONSECC	Out	60%	Security level asserted to indicate a non-secure transfer
HREADYC	In	60%	Slave ready
HRDATAC[31:0]	In	60%	Read data
HRESPC	In	60%	Slave response
HMASTERC	Out	60%	Initiator of the Transfer  0 = Processor  1 = Debugger
HEXCLC	Out	60%	Exclusive request  Address phase control signal that indicates whether an access is because of a LDREX or STREX instruction  0 = non-exclusive (standard) transaction  1 = exclusive transaction



Signal name	Direction	Timing	Description
HEXOKAYC	In	60%	<p>Exclusive response</p> <p>Data phase signal sampled on HREADYC that indicates whether the exclusive request was granted or not</p> <p>1 = exclusive access granted</p> <p>0 = exclusive access failed</p>

### C.10.1 CODE-AHB interface protection signals

The following signals are included to provide additional protection for the interface in Functional Safety applications.

**Table C-12: CODE-AHB interface protection signals**

Signal name	Direction	Timing	Description
HREADYCHKC	In	60%	Odd parity of HREADYC
HTRANSCHKC	Out	50%	Odd parity of HTRANS[1:0]
HADDRCHKC[3:0]	Out	60%	Odd parity of HADDR[31:0] at 8-bit granularity
HRDATACHKC[3:0]	In	60%	Odd parity of HRDATA[31:0] at 8-bit granularity
HWDATACHKC[3:0]	Out	60%	Odd parity of HWDATA[31:0] at 8-bit granularity
HPROTCHKC	Out	60%	Odd parity of HPROTC[6:0]
HCTRLCHK1C	Out	60%	Odd parity of {HBURSTC[2:0], HNONSECC, HWRITEC, HSIZEC[2:0]}
HCTRLCHK2C	Out	60%	Odd parity of {HEXCLC, HMASTERC}
HRESPCHKC	In	60%	Odd parity of {HRESPC, HEXOKAYC}

## C.11 SYS-AHB interface signals

The SYS-AHB interface implements the standard set of AMBA AHB signals. Cortex®-M52 can be configured to use either AXI interface or AHB interface. If AHB interface is used, then the CODE-AHB interface and SYS-AHB interface will be used instead of the AXI interface.

The following table shows the SYS-AHB interface signals.

**Table C-13: SYS-AHB interface signals**

Signal name	Direction	Timing	Description
HTRANSYS[1:0]	Out	50%	Transfer type
HBURSTSYS[2:0]	Out	60%	Transfer burst length
HADDRSYS[31:0]	Out	60%	Transfer address
HWRITESYS	Out	60%	Write transfer
HSIZESYS[2:0]	Out	60%	Transfer size
HWDATASYS[31:0]	Out	60%	Write data
HPROTSYS[6:0]	Out	60%	Protection and outer memory attributes
HNONSECSYS	Out	60%	Security level asserted to indicate a non-secure transfer
HREADYSYS	In	60%	Slave ready
HRDATASYS[31:0]	In	60%	Read data
HRESPSYS	In	60%	Slave response
HMASTERSYS	Out	60%	Initiator of the Transfer. <ul style="list-style-type: none"> <li>0 = Processor</li> <li>1 = Debugger</li> </ul>
HEXCLSYS	Out	60%	Exclusive request  Address phase control signal that indicates whether an access is because of a LDREX or STREX instruction. <ul style="list-style-type: none"> <li>0 = non-exclusive (standard) transaction</li> <li>1 = exclusive transaction</li> </ul>
HEXOKAYSYS	In	60%	Exclusive response  Data phase signal sampled on HREADYC that indicates whether the exclusive request was granted or not. <ul style="list-style-type: none"> <li>1 = exclusive access granted</li> <li>0 = exclusive access failed</li> </ul>

### C.11.1 SYS-AHB interface protection signals

The following signals are included to provide additional protection for the interface in Functional Safety applications

**Table C-14: SYS-AHB interface protection signals**

Signal name	Direction	Timing	Description
HREADYCHSYS	In	60%	Odd parity of HREADYSYS

Signal name	Direction	Timing	Description
HTRANSCHKSYS	Out	50%	Odd parity of HTRANSYS[1:0]
HADDRCHKSYS[3:0]	Out	60%	Odd parity of HADDRSYS[31:0] at 8-bit granularity
HRDATACHKSYS[3:0]	In	60%	Odd parity of HRDATASYS[31:0] at 8-bit granularity
HWDATACHKSYS[3:0]	Out	60%	Odd parity of HWDATASYS[31:0] at 8-bit granularity
HPROTCHKSYS	Out	60%	Odd parity of HPROTSYS[6:0]
HCTRLCHK1SYS	Out	60%	Odd parity of {HBURSTSYS[2:0], HNONSECSYS, , HWWRITESYSHSIZESYS[2:0]}
HCTRLCHK2SYS	Out	60%	Odd parity of {HEXCLSYS, HMASTERSYS}
HRESPCHKSYS	In	60%	Odd parity of {HRESPSYS, HEXOKAYSYS}

## C.12 TCM-AHB interface signals

The TCM-AHB interface provides direct access to the processor *Tightly Coupled Memory* (TCM) interfaces.

The following table shows the signals for the TCM-AHB interface.

**Table C-15: TCM-AHB interface signals**

Signal name	Direction	Description
HSELS	Input	This signal selects access to <i>Tightly Coupled Memory</i> (TCM) interfaces.
HTRANS[1:0]	Input	Transfer type.
HBURSTS[2:0]	Input	Transfer burst length.
HADDRS[31:0]	Input	Transfer address and selected TCM interface.
HWWRITES	Input	Write transfer.
HSIZES[2:0]	Input	Transfer size.
HWDATAS[31:0]	Input	Write data.
HWSTRBS[3:0]	Input	Write data byte lane strobes.
HPROTS[6:0]	Input	Protection and outer memory attributes.
HNONSECS	Input	Security level, asserted to indicate a Non-secure transfer. For more information, see the <i>AMBA® 5 AHB Protocol Specification</i> .
HREADY	Input	Data phase that is associated with the previous transfer on the interconnect is complete. The interconnect sends the signal to all AHB slaves and to the master, which started the transfer.
HREADYOUTS	Output	Slave ready.
HRDATAS[31:0]	Output	Read data.
HRESP	Output	Slave response.

Signal name	Direction	Description
SAHBWABORT	Output	Indicates asynchronous abort for writes from TCM errors indicated on ITCMERR, D0TCMERR, or D1TCMERR.

## C.12.1 TCM-AHB interface protection signals

The following table shows the *AHB TCM Access* (TCM-AHB) interface protection signals.

**Table C-16: TCM-AHB interface protection signals**

Signal name	Direction	Description
HREADYCHKS	Input	Odd parity of HREADYS.
HREADYOUTCHKS	Output	Odd parity of HREADYOUTS.
HTRANSCHKS	Input	Odd parity of HTRANS[1:0].
HADDRCHKS[3:0]	Input	Odd parity of HADDRS[31:0] at 8-bit granularity.
HRDATACHKS[3:0]	Output	Odd parity of HRDATAS[31:0] at 8-bit granularity.
HWDATACHKS[3:0]	Input	Odd parity of HWDATA[31:0] at 8-bit granularity.
HWSTRBCHKS	Input	Odd parity of HWSTRBS[3:0].
HPROTCHKS	Input	Odd parity of HPROTS[6:0].
HCTRLCHK1S	Input	Odd parity of (HBURSTS[2:0], HNONSECS, HWWRITES, HSIZE[2:0])
HRESPCHKS	Output	Odd parity of HRESPS.
HSELCHKS	Input	Odd parity of HSELS.
SAHBWABORTCHK	Output	Odd parity of SBAHBWABORT

## C.13 P-AHB interface signals

The *Peripheral AHB* (P-AHB) interface implements the standard set of AMBA® 5 AHB signals.

The following table shows the signals for the P-AHB interface.

**Table C-17: P-AHB interface signals**

Signal name	Direction	Description
HTRANS[1:0]	Output	Transfer type.
HBURST[2:0]	Output	Transfer burst length.
HADDR[31:0]	Output	Transfer address.
HWRITEP	Output	Write transfer.
HSIZE[2:0]	Output	Transfer size.
HWDATAP[31:0]	Output	Write data.
HPROTP[6:0]	Output	Protection and outer memory attributes.  <b>Note:</b> HPROTP[0] is always 0b1 as the interface does not support instruction fetch.

Signal name	Direction	Description
HNONSECP	Output	Asserted to indicate a Non-secure transfer.
HREADYP	Input	Slave ready.
HRDATAP[31:0]	Input	Read data.
HRESPP	Input	Slave response.
HMASTERP	Output	Initiator of the access:  <b>0</b> Processor access. <b>1</b> Debugger access.
HEXCLP	Output	Exclusive request.  Address phase control signal that indicates whether an access is a result of either a: <ul style="list-style-type: none"> <li>LDREX instruction.</li> <li>STREX instruction.</li> </ul> <b>0</b> Non-exclusive (standard) transaction. <b>1</b> Exclusive transaction.
HEXOKAYP	Input	Exclusive response.  This data phase signal is sampled on HREADYP, and it indicates whether the exclusive request was granted.  <b>0</b> Exclusive access failed. <b>1</b> Exclusive access that is granted.

### C.13.1 P-AHB interface protection signals

The following table shows the *Peripheral AHB* (P-AHB) interface protection signals.

**Table C-18: P-AHB interface protection signals**

Signal name	Direction	Description
HREADYCHKP	Input	Odd parity of HREADYP.
HTRANSCHKP	Output	Odd parity of HTRANSP[1:0].
HADDRCHKP[3:0]	Output	Odd parity of HADDRP[31:0] at 8-bit granularity.
HRDATACHKP[3:0]	Input	Odd parity of HRDATAP[31:0] at 8-bit granularity.
HWDATACHKP[3:0]	Output	Odd parity of HWDATAP[31:0] at 8-bit granularity.
HCTRLCHK1P	Output	Odd parity of (HBURSTP[2:0], HNONSECP, HWRITEP, HSIZEP[2:0])
HCTRLCHK2P	Output	Odd parity of (HEXCLP, HMASTERP)
HPROTCHKP	Output	Odd parity of HPROTP[6:0].
HRESPCHKP	Input	Odd parity of (HRESPP, HEXOKAYP)

## C.14 D-AHB interface signals

The following table shows the *Debug AHB* (D-AHB) interface signals.

**Table C-19: D-AHB interface signals**

Signal name	Direction	Description
HTRANS[1:0]	Input	Indicates the type of current transfer.  <b>Note:</b> HTRANS[0] is ignored by the processor, all transactions are treated as either Non-sequential or Idle.
HBURST[2:0]	Input	Transfer burst length. Indicates whether the transfer is part of a burst. Debug accesses are always treated as SINGLE, and this signal is ignored.
HADDR[31:0]	Input	Transfer address.
HWRITED	Input	Write transfer.
HSIZE[2:0]	Input	Transfer size. Indicates the size of the access. Accesses can be:  <div> <div>0b000</div> <div>0b001</div> <div>0b010</div> </div> <div> <div>Byte.</div> <div>Halfword.</div> <div>Word.</div> </div> <b>Note:</b> HSIZE[2] is ignored by the processor.
HWDAT[31:0]	Input	Write data. Data write bus.
HPROT[6:0]	Input	Protection and outer memory attributes. Provides information on the access.  <b>Note:</b> HPROT[0] is ignored by the processor, all debug transactions are treated as data accesses.
HNONSEC	Input	Security level that is requested by debug access, asserted to indicate a Non-secure transfer. The resultant security level of the debug access depends on the debug control registers in the processor and the debug access control signals.
HREADY	Output	Slave ready. When HIGH indicates that a transfer has completed on the bus. This signal is driven LOW to extend a transfer.
HRDAT[31:0]	Output	Read data.
HRESP	Output	Slave response

### C.14.1 D-AHB interface protection signals

The following table shows the *Debug AHB* (D-AHB) interface signals.

**Table C-20: D-AHB interface protection signals**

Signal name	Direction	Description
HREADYCHK	Output	Odd parity of HREADY.
HTRANSCHK	Input	Odd parity of HTRANS[1:0].
HADDRCHK[3:0]	Input	Odd parity of HADDR[31:0] at 8-bit granularity.

Signal name	Direction	Description
HRDATACHKD[3:0]	Output	Odd parity of HRDATAD[31:0] at 8-bit granularity.
HWDATACHKD[3:0]	Input	Odd parity of HWDATAD[31:0] at 8-bit granularity.
HCTRLCHK1D	Input	Odd parity of (HBURSTD[2:0], HNONSECD, HWRITED, HSIZED[2:0]).
HPROTCHKD	Input	Odd parity of HPROTD[6:0].
HRESPCHKD	Output	Odd parity of HRESPD.

## C.15 EPPB interface signals

The following table shows the *External Private Peripheral Bus* (EPPB) APB interface signals.

**Table C-21: EPPB signals**

Signal name	Direction	Description
PSEL	Output	APB device select. Indicates that a data transfer is requested.
PENABLE	Output	APB control signal. Strobe to time all accesses. Indicates the access phase of an APB transfer.
PPROT[2:0]	Output	Transfer privilege and security level.
PWRITE	Output	Write transfer.
PSTRB[3:0]	Output	Write data byte strobes
PADDR[19:2]	Output	Transfer address.
PADDR31	Output	Initiator of the transfer.  <b>0</b> Processor <b>1</b> Debugger
PWDATA[31:0]	Output	APB 32-bit write data bus.
PREADY	Input	APB slave ready signal. This signal is driven LOW if the currently accessed APB device requires extra wait states to complete the transfer.
PSLVERR	Input	APB slave error signal. This signal is driven HIGH if the currently accessed APB device cannot handle the requested transfer.
PRDATA[31:0]	Input	APB 32-bit read data bus.

### C.15.1 EPPB interface protection signals

The following table shows the *External Peripheral Bus* (EPPB) interface protection signals.

**Table C-22: EPPB interface protection signals**

Signal name	Direction	Description
PSELCHK	Output	Odd parity of PSEL.
PREADYCHK	Input	Odd parity of PREADY.
PENABLECHK	Output	Odd parity of PENABLE.
PADDRCHK[3:0]	Output	Odd parity, at 8-bit granularity, of (PADDR31, 0b000000000000, PADDR[19:2], 0b00)
PRDATACHK[3:0]	Input	Odd parity of PRDATA[31:0] at 8-bit granularity.

Signal name	Direction	Description
PWDATACHK[3:0]	Output	Odd parity of PWDATA[31:0] at 8-bit granularity.
PCTRLCHK	Output	Odd parity of (PPROT[2:0],PWRITE)
PSTRBCHK	Output	Odd parity of PSTRB[3:0].
PSLVERRCHK	Input	Odd parity of PSLVERR.

## C.16 External coprocessor interface signals

The following table lists the external coprocessor interface signals.

**Table C-23: External coprocessor interface signals**

Signal name	Direction	Description
CPRESETOUTn	Output	This signal is asserted when the processor PDCORE domain is in reset.
CPENABLED[7:0]	Output	Indicates which coprocessor is enabled in the: <ul style="list-style-type: none"> <li>CPACR register associated with the Security state of the processor.</li> <li>NSACR register if the processor is executing in Non-secure state.</li> </ul> <p><b>Note:</b> The CPACR is banked when the implementation includes the Security Extension.</p>
CPPWRSU[7:0]	Output	Indicates which coprocessors are permitted to become <b>UNKNOWN</b> .
CPSPRESENT[7:0]	Input	Indicates which Secure coprocessors are present in the system.
CPNSPRESENT[7:0]	Input	Indicates which Non-secure coprocessors are present in the system.
CPCDP	Output	Coprocessor command operation.
CPMCR	Output	Coprocessor register transfer from processor operation.
CPMRC	Output	Coprocessor register transfer to processor operation.
CPSIZE	Output	Coprocessor size operation.
CPNUM[2:0]	Output	Coprocessor number request.
CPREGS[11:0]	Output	Operation register fields.
CPOPC[8:0]	Output	Operation opcode fields.
CPPRIV	Output	Indicates operation privilege.
CPNSATTR	Output	Indicates operation Security state.
CPVALID	Output	Indicates whether the coprocessor operation is valid.
CPREADY	Input	Indicates whether the coprocessor is stalled or ready.
CPERROR	Input	Indicates that the coprocessor is not present or the instruction is not supported.
CPWDATA[63:0]	Output	The coprocessor write data bus.
CPRDATA[63:0]	Input	The coprocessor read data bus.



## C.17 Arm Custom Instructions signals

The Cortex®-M52 processor implements Arm Custom Instruction (ACI) through the Custom Datapath Extension (CDE) for Armv8-M.

### ACI signals

The ACI signals for the CDE and EPCDE modules are documented in the *Cortex®-M52 Processor Integration and Implementation Manual*, which is a confidential document available only to licensees.

The following table describes the ACI static configuration signal.

**Table C-24: ACI static configuration signal**

Signal name	Direction	Description
CFGNOCDCEP[7:0]	Input	Disable support for mapping of CDE onto coprocessor instructions.  If Verilog parameter CDEMAPPEDONCPn is set to 1, setting CFGNOCDCEP[n] to 1 will force the external interface coprocessor instruction behaviour for CPn.

## C.18 Debug interface signals

The following table shows the debug interface signals.



For more information on debug authentication, see the section on authentication rules in the *Arm® CoreSight™ Architecture Specification v3.0*.

**Table C-25: Debug signals**

Signal name	Direction	Description
HALTED	Output	In halting mode debug. HALTED remains asserted while the processor is in debug.
DBGRESTART	Input	Request for synchronized exit from halt mode. Forms a handshake with DBGRESTARTED. If multiprocessor debug support is not required, DBGRESTART must be tied LOW.
DBGRESTARTED	Output	Handshake for DBGRESTART.
EDBGRQ	Input	External debug request. A debug agent in the system asserts this signal to request that the processor enters Debug state.
DBGEN	Input	Invasive debug enable. When LOW, disables all halt-mode and invasive debug features.
NIDEN	Input	Non-invasive debug enable. When LOW, disables all trace and non-invasive debug features.
SPIDEN	Input	Secure invasive debug enable. When LOW, disables all halt mode and invasive debug features when the processor is in Secure state.
SPNIDEN	Input	Secure non-invasive debug enable.  Controls access to non-invasive debug features when the processor is in Secure state and SPIDEN is LOW.

## C.19 P-Channel and Q-Channel power control signals

The Cortex®-M52 processor PDCORE and PDRAMS power domains are controlled by a P-Channel interface because there are multiple power modes, and each power mode is a combination of states for these domains. The debug power domain, PDDEBUG, is controlled by a Q-Channel interface because there are only two power modes, that is, ON and OFF.

### PDCORE P-Channel interface signals

The following table shows the PDCORE and PDRAMS P-Channel signals.



- For applications using `LOCKSTEP`, the `COREPREQ` signal must be synchronized outside the processor. This ensures that both processors in lockstep received `COREPREQ` changes on the same cycle.
- For more information on `COREPACTIVE` signal encoding, see [COREPACTIVE signal encoding](#).

### PDCORE P-Channel interface signals

**Table C-26:**

Signal name	Direction	Description
COREPREQ	Input	Request to transition to power mode indicated by COREPSTATE.
COREPSTATE[4:0]	Input	Requested power mode.
COREPACCEPT	Output	Acceptance of the transition to the requested power mode.
COREPDENY	Output	Denial of the power mode transition request.
COREPACTIVE[20:0]	Output	Hint signal from processor for minimum required mode.

### PDDEBUG Q-Channel interface signals

The following table shows the PDDEBUG Q-Channel interface signals.



The Q-Channel input `PWRDBGQREQn` signal is asynchronous to `DBGCLK` Cortex®-M52 processor.

**Table C-27: PDDEBUG Q-Channel interface signals**

Signal name	Direction	Description
PWRDBGQREQn	Input	Debug domain quiescence request signal.
PWRDBGQACCEPTn	Output	Debug domain quiescence request accepted, and is synchronized inside the
PWRDBGQDENY	Output	Debug domain quiescence request denied.
PWRDBGQACTIVE	Output	Debug logic active or activation request.
PWRDBGWAKEQACTIVE	Output	Debug request in progress. System-level power control must power up PDDEBUG domain to complete transaction.

## C.20 Q-Channel clock control signals

The CLKIN and DBGCLK, which can be gated at the system-level, is controlled by a separate Q-Channel interface.

The following table shows the Q-Channel signals for CLKIN clock control.



The Q-Channel input CLKINQREQn signal is asynchronous to CLKIN and is synchronized inside the Cortex®-M52 processor.

**Table C-28: Q-Channel for CLKIN control**

Signal name	Direction	Description
CLKINQREQn	Input	Q-Channel for CLKIN control.
CLKINCLKQACCEPTn	Output	
CLKINQDENY	Output	
CLKINQACTIVE	Output	

The following table shows the debug Q-Channel signals for DBGCLK clock control.



The Q-Channel input DBGCLKQREQn signal is asynchronous to DBGCLK and is synchronized inside the Cortex®-M52 processor.

**Table C-29: Q-Channel signals for DBGCLK control**

Signal name	Direction	Description
DBGCLKQREQn	Input	Q-Channel for DBGCLK clock control.
DBGCLKQACCEPTn	Output	
DBGCLKQDENY	Output	
DBGCLKQACTIVE	Output	

## C.21 Power compatibility control signals

The following table shows the power compatibility control signals.

**Table C-30: Power compatibility control signals**

Signal name	Direction	Description
SLEEPING	Output	When HIGH indicates that the processor is ready to enter a low-power state. When LOW, indicates that the processor is running or wants to leave sleep mode.  If SLEEPHOLDACKn is LOW, then the processor does not perform any fetches until SLEEPHOLDREQn is driven HIGH.
SLEEPDEEP	Output	Indicates that the processor and ETM are ready to enter a low-power state and the wake up time is not critical. Only active when SLEEPING is HIGH.
SLEEPHOLDACKn	Output	Acknowledge signal for SLEEPHOLDREQn. If this signal is LOW, irrespective of the SLEEPING signal value, the processor does not advance in execution and does not perform any memory operations.
SLEEPHOLDREQn	Input	Request to extend the processor sleeping state regardless of wake up events. If the processor acknowledges this request driving SLEEPHOLDACKn LOW, this guarantees the processor remains idle even when receiving a wake up event.

## C.22 ITM interface signals

The following table shows the ATB *Instrumentation Trace Macrocell* (ITM) interface signals.

**Table C-31: ITM interface signals**

Signal name	Direction	Description
AFREADYI	Output	Trace flush acknowledge.
AFVALIDI	Input	Trace flush request.
ATDATAI[7:0]	Output	Trace data.
ATIDI[6:0]	Output	Trace source ID.
ATREADYI	Input	Trace slave ready.
ATVALIDI	Output	Trace transfer valid.
SYNCREQI	Input	ITM trace synchronization request.

## C.23 ETM interface signals

The following table shows the ATB CoreSight™ *Embedded Trace Macrocell* (ETM) trace interface signals.

**Table C-32: ETM interface signals**

Signal name	Direction	Description
ATVALIDE	Output	Trace transfer is valid.
ATIDE[6:0]	Output	Trace source ID.

Signal name	Direction	Description
ATDATAE[7:0]	Output	Trace data.
AFREADYE	Output	Trace flush acknowledge.
AFVALIDE	Input	Trace flush request.
ATREADYE	Input	Trace slave is ready.
SYNCREQE	Input	ETM Trace synchronization request.

## C.24 Trace synchronization and trigger signals

The following table shows the trace synchronization and trigger interface signals

**Table C-33: Trace synchronization and trigger signals**

Name	Type	Description
TRCENA	Output	Status of the DEMCR.TRCENA register, indicating whether the <i>Data Watchpoint Trace</i> (DWT) and <i>Instrumentation Trace Macrocell</i> (ITM) units are enabled (when implemented).
TPIUACTV	Input	TPIU data active.
TPIUBAUD	Input	TPIU Baud indicator
DSYNC	Output	DWT synchronization request.
ETMTRIGOUT	Output	ETM trigger event output bit[0]. Indicates a trigger packet in the trace stream.

## C.25 CTI interface signals

The following table shows the *Cross Trigger Interface* (CTI) interface signals.

**Table C-34: CTI signals**

Signal name	Direction	Description
CTICHIN[3:0]	In	CTI channel input
CTICHOUT[3:0]	Out	CTI channel output
CTIIRQ[1:0]	Out	CTI interrupt request

## C.26 Interrupt signals

All interrupt inputs must be generated synchronously to CLKIN. Both pulse and level interrupts are supported.

The following table shows the interrupt signals

**Table C-35: Interrupt signals**

Signal name	Direction	Description				
IRQ[479:0]	Input	<p>External interrupt signals. The NUMIRQ parameter configures the implemented bits of this signal.</p> <p><b>Note:</b></p> <ul style="list-style-type: none"><li>IRQ and NMI signals are active-HIGH and the hardware is agnostic between pulse- and level-signaled interrupts.</li><li>You must ensure that the IRQ and NMI signals to the processor are synchronized to CLKIN using the appropriate circuit.</li></ul>				
NMI	Input	Non-maskable interrupt				
CURRPRI[7:0]	Output	<p>Current interrupt priority level.</p> <p>If the processor is in Handler mode for an exception with configurable priority CURRPRI indicates the programmed priority level of the exception.</p> <p>If the processor is in handler mode for an exception with negative priority CURRPRI is 0.</p> <p>If the processor is in Thread mode CURRPRI is dependent on whether a base priority mask is enabled by setting BASEPRI &gt; 0:</p> <table><tr><td><b>BASEPRI==0</b></td><td>CURRPRI=0</td></tr><tr><td><b>BASEPRI &gt; 0</b></td><td>CURRPRI=BASEPRI</td></tr></table> <p>The current exception number can be determined using the output signal INTNUM.</p>	<b>BASEPRI==0</b>	CURRPRI=0	<b>BASEPRI &gt; 0</b>	CURRPRI=BASEPRI
<b>BASEPRI==0</b>	CURRPRI=0					
<b>BASEPRI &gt; 0</b>	CURRPRI=BASEPRI					
INTNUM[8:0]	Output	<p>Interrupt number of the current execution context, from bits [8:0] of IPSR.</p> <p><b>Note:</b></p> <ul style="list-style-type: none"><li>When the processor is in Thread mode, INTNUM is 0.</li><li>When the processor is in Handler mode, INTNUM is the exception number of the currently executing exception.</li></ul>				

## C.27 WIC interface signals

There are two *Wakeup Interrupt Controller* (WIC) units that the processor supports.

- The *Internal Wakeup Interrupt Controller* (IWIC) that is present inside the processor.
- The *External Wakeup Interrupt Controller* (EWIC) that is an external peripheral to the processor.

### WIC configuration signal

The following table shows the WIC configuration signal.

**Table C-36: WIC configuration signal**

Signal name	Direction	Description
WICCONTROL[3:0]	Input	<p>This signal is responsible for WIC control and configuration.</p> <p><b>WICCONTROL[3]</b> This bit indicates the EWIC automatic sequence on powerdown sequence. This bit is connected to EWIC in the system.</p> <p><b>WICCONTROL[2]</b> This bit indicates the EWIC automatic sequence on powerup sequence. This bit is connected to EWIC in the system.</p> <p><b>WICCONTROL[1]</b> This bit indicates that IWIC must be used.</p> <p><b>WICCONTROL[0]</b> This bit indicates that SLEEPDEEP is WIC sleep.</p> <p><b>Note:</b></p> <ul style="list-style-type: none"> <li>If No Wakeup interrupt controller is included, WICCONTROL[3:0] must be tied to 0b0000.</li> <li>If the IWIC is not included in the processor configuration, WICCONTROL[1] must be tied to 0b0.</li> </ul> <p>If the EWIC is not included in the system, WICCONTROL[3:1] must be tied to 0b001.</p>

## IWIC interface signals

The following table shows the IWIC signals.

**Table C-37: IWIC signals**

Signal name	Direction	Description								
IWICCLK	Input	This signal is the IWIC clock.								
nIWICRESET	Input	<p>This is an active-LOW IWIC reset signal. This signal is internally synchronized to IWICCLK. If Dual-Core Lockstep is not configured in the processor the nIWICRESET signal is treated as an asynchronous input. Reset assertion is fully asynchronous and does not require an active clock. Reset de-assertion is synchronised inside the processor.</p> <p>If DCLS is configured in the processor, by setting the Verilog parameter LOCKSTEP, this signal must be asserted and de-asserted together with nIWICRESETDCLS. If IWICCLK is active when nIWICRESET is asserted or de-asserted, then the signal must be constrained such that nIWICRESET is stable on the rising edge of the clock.</p>								
IWAKEUP	Output	This signal indicates the IWIC wake-up event that is detected when the processor is in WIC sleep.								
IWICSENSE[482:0]	Output	<p>This signal indicates which input events cause the WIC to generate the IWAKEUP signal. The WICLINES configuration parameter determines the usable width of this signal. Therefore, only the IWICSENSE[WICLINES-1:0] bits are implemented and the remaining bits are driven LOW.</p> <p>The mapping to input events is:</p> <table><tr><td><b>IWICSENSE[482:3]</b></td><td>IRQ[479:0].</td></tr><tr><td><b>IWICSENSE[2]</b></td><td>EDBGRQ.</td></tr><tr><td><b>IWICSENSE[1]</b></td><td>NMI.</td></tr><tr><td><b>IWICSENSE[0]</b></td><td>RXEV.</td></tr></table> <p><b>Note:</b></p> <ul style="list-style-type: none"><li>• If No Wakeup interrupt controller is included, WICCONTROL[3:0] must be tied to 0b0000.</li><li>• If the IWIC is not included in the processor configuration, WICCONTROL[1] must be tied to 0b0.</li></ul> <p>If the EWIC is not included in the system, WICCONTROL[3:1] must be tied to 0b001.</p>	<b>IWICSENSE[482:3]</b>	IRQ[479:0].	<b>IWICSENSE[2]</b>	EDBGRQ.	<b>IWICSENSE[1]</b>	NMI.	<b>IWICSENSE[0]</b>	RXEV.
<b>IWICSENSE[482:3]</b>	IRQ[479:0].									
<b>IWICSENSE[2]</b>	EDBGRQ.									
<b>IWICSENSE[1]</b>	NMI.									
<b>IWICSENSE[0]</b>	RXEV.									

## EWIC interface signal

The following table shows the EWIC signal.

**Table C-38: EWIC signal**

Signal name	Direction	Description
EWAKEUP	Input	The processor uses this signal to drive the COREACTIVE output signal. This signal is asserted to indicate when a wakeup event is detected in WIC sleep. For more information on COREACTIVE, see <a href="#">P-Channel and Q-Channel power control signals</a> .

## C.28 Event signals

The following table shows the event signals.

**Table C-39: Event signals**

Signal name	Direction	Description
TXEV	Output	This signal is a notification of an event that the processor generates when the SEV instruction is executed. This signal is a single-cycle pulse signal.
RXEV	Input	This signal is a notification of a system event.
LOCKUP	Output	This signal is a notification that the processor is in the architected lockup state because of an unrecoverable exception.
EVENTBUS[223:0]	Output	This signal indicates the <i>Performance Monitoring Unit</i> (PMU) events. EVENTBUS[n] is pulsed for a single cycle for each event, n, on the processor.

## C.29 IDAU interface signals

An *Implementation Defined Attribution Unit* (IDAU) can control the security attributes for most of the memory the Cortex®-M52 processor addresses to a granularity of 32 bytes.

The following table shows the IDAU interface signals.

**Table C-40: IDAU interface signals**

Signal Name	Direction	Description
IDAUVALIDA	Output	Port A address valid
IDAUADDRA[31:5]	Output	Port A address
IDAUVALIDB	Output	Port B address valid
IDAUADDRB[31:5]	Output	Port B address
IDAUNSA	Input	Port A Non-secure
IDAUNSCA	Input	Port A Non-secure Callable
IDAUNSB	Input	Port B Non-secure
IDAUNSCB	Input	Port B Non-secure Callable
IDAUIDA[7:0]	Input	Port A region number



Signal Name	Direction	Description
IDAUIDB[7:0]	Input	Port B region number
IDAUDVA	Input	Port A region number valid
IDAUDVB	Input	Port B region number valid
IDAUNCHKA	Input	Port A region exempt from attribution check
IDAUNCHKB	Input	Port B region exempt from attribution check

## C.30 Miscellaneous signals

The following table shows the miscellaneous signals. The configuration input signals are sampled at reset.

**Table C-41: Miscellaneous interface signals**

Signal name	Direction	Description
TSVALUEB[63:0]	Input	Binary coded global timestamp count. This signal is synchronous to CLKIN.
TSCLKCHANGE	Input	This signal indicates timestamp clock ratio change.
SYSRESETREQ	Output	Request for functional reset. This can be done using either nSYSRESET or a combination of the P-Channel interface and nSYSRESET.
ECOREVNUM[35:0]	Input	ECO revision number. The ECO revision field mappings are:  <div> <div>[35:32]</div> <div>[31:28]</div> <div>[27:24]</div> <div>[23:20]</div> <div>[15:12]</div> <div>[11:8]</div> <div>[7:4]</div> <div>[3:0]</div> </div> <div> <div>Performance Monitoring Unit (PMU)</div> <div>Embedded Trace Macrocell (ETM).</div> <div>Cross Trigger Interface (CTI).</div> <div>ROM table.</div> <div>Instrumentation Trace Macrocell (ITM).</div> <div>System Control Space (SCS).</div> <div>Data Watchpoint and Trace (DWT).</div> <div>[19:16]BreakPoint Unit (BPU).</div> <div>CPUID revision.</div> </div>
REVIDRNUM[3:0]	Input	Revision ID Number  This value of this signal is reflected in the Revision ID register REVIDR[3:0].

Signal name	Direction	Description
LOCKSVTAIRCR	Input	<p>Disables writes to the following secure registers from software or from a debug agent that is connected to the processor.</p> <ul style="list-style-type: none"> <li>• VTOR_S.</li> <li>• AIRCR.PRIS.</li> <li>• AIRCR.BFHFNMINS.</li> </ul> <p>Asserting this signal:</p> <ul style="list-style-type: none"> <li>• Prevents changes to the secure vector table base address.</li> <li>• Handling of secure interrupt priority.</li> <li>• Handling of BusFault, HardFault, and NMI security target settings in the processor.</li> </ul> <p>For more information on these registers, see the <i>Arm®v8-M Architecture Reference Manual</i>.</p> <p>This signal can be changed dynamically.</p> <p>When SECEXT=0, VTOR_S and associated signals exist but do not have any effect, and only VTOR_NS and its associated signals exist.</p>
LOCKNSVTOR	Input	<p>Disables writes to the VTOR_NS register.</p> <p>For more information on this register, see <i>Arm®v8-M Architecture Reference Manual</i>.</p> <p>Asserting this signal prevents changes to the Non-secure vector table base address.</p> <p>This signal can be changed dynamically.</p> <p>When SECEXT=0, VTOR_S and associated signals exist but do not have any effect, and only VTOR_NS and its associated signals exist.</p>
LOCKSMPU	Input	<p>This signal disables writes to registers that are associated with the Secure Memory Protection Unit (MPU) region from software or from a debug agent connected to the processor.</p> <ul style="list-style-type: none"> <li>• MPU_CTRL.</li> <li>• MPU_RNR.</li> <li>• MPU_RBAR.</li> <li>• MPU_RLAR.</li> <li>• MPU_RBAR_An.</li> <li>• MPU_RLAR_An.</li> </ul> <p>For more information on these registers, see the <i>Arm®v8-M Architecture Reference Manual</i>.</p> <p>Asserting this signal prevents changes to the memory regions which have been programmed in the secure MPU. All writes to the registers are ignored.</p> <p>This signal has no effect if the Cortex®-M52 processor has not been configured with support for the Security Extension, or if no Secure MPU regions have been configured.</p> <p>This signal can be changed dynamically.</p>

Signal name	Direction	Description
LOCKNSMPU	Input	<p>This signal disables writes to registers that are associated with the Non-secure MPU region from software or from a debug agent connected to the processor.</p> <ul style="list-style-type: none"> <li>MPU_CTRL_NS.</li> <li>MPU_RNR_NS.</li> <li>MPU_RBAR_NS.</li> <li>MPU_RLAR_NS.</li> <li>MPU_RBAR_A_NSn.</li> <li>MPU_RLAR_A_NSn.</li> </ul> <p>For more information on these registers, see the <i>Arm®v8-M Architecture Reference Manual</i>.</p> <p>Asserting this signal prevents changes to the memory regions which have been programmed in the Non-secure MPU. All writes to the registers are ignored.</p> <p>This signal has no effect if the Cortex®-M52 processor has not been configured with support for Non-secure MPU regions.</p> <p>This signal can be changed dynamically.</p>
LOCKSAU	Input	<p>This signal disables writes to registers that are associated with the <i>Security Attribution Unit</i> (SAU) region from software or from a debug agent connected to the processor.</p> <ul style="list-style-type: none"> <li>SAU_CTRL.</li> <li>SAU_RNR.</li> <li>SAU_RBAR.</li> <li>SAU_RLAR.</li> </ul> <p>For more information on these registers, see the <i>Arm®v8-M Architecture Reference Manual</i>.</p> <p>Asserting this signal prevents changes to the memory regions which have been programmed in the SAU. All writes to the registers are ignored.</p> <p>This signal has no effect if the Cortex®-M52 processor has not been configured with support for the Security Extension, or if no SAU regions have been configured.</p> <p>This signal can be changed dynamically.</p>
LOCKTCM	Input	<p>This signal disables writes to registers that are associated with the TCM region from software or from a debug agent connected to the processor.</p> <ul style="list-style-type: none"> <li>ITCMCR.</li> <li>DTCMCR.</li> </ul> <p>For more information on these registers, see the <i>Arm® Cortex®-M52 Processor Technical Reference Manual</i>.</p> <p>Asserting this signal prevents changes to the TCM configuration. All writes to the registers are ignored.</p>

Signal name	Direction	Description
LOCKITGU	Input	<p>This signal disables writes to registers that are associated with the ITCM interface security gating from software or from a debug agent connected to the processor.</p> <ul style="list-style-type: none"> <li>ITGUCTRL.</li> <li>ITGU_LUTn.</li> </ul> <p>For more information on these registers, see the <i>Arm® Cortex®-M52 Processor Technical Reference Manual</i>.</p> <p>Asserting this signal prevents changes to the security gating configuration of the ITCM.</p>
LOCKDTGU	Input	<p>This signal disables writes to registers that are associated with the DTCM interface security gating from software or from a debug agent connected to the processor.</p> <ul style="list-style-type: none"> <li>DTGUCTRL.</li> <li>DTGU_LUTn.</li> </ul> <p>For more information on these registers, see the <i>Arm® Cortex®-M52 Processor Technical Reference Manual</i>.</p> <p>Asserting this signal prevents changes to the security gating configuration of the DTCM.</p>
LOCKPAHB	Input	<p>Disable writes to the PAHBPCR register from software or from a debug agent connected to the processor.</p> <p>For more information on this register, see the <i>Arm® Cortex®-M52 Processor Technical Reference Manual</i>.</p> <p>Asserting this signal prevents changes to P-AHB port enable status in PAHBPCR.EN.</p>
LOCKDCAIC	Input	<p>Disable access to the instruction cache direct cache access registers DCAICLR and DCAICRR.</p> <p>Asserting this signal prevents direct access to the instruction cache Tag or Data RAM content. This is required when using <i>eXecutable Only Memory (XOM)</i> on the M-AXI or M-AHB master interface.</p> <p>When LOCKDCAIC is asserted:</p> <ul style="list-style-type: none"> <li>DCAICLR is RAZ/WI.</li> <li>DCAICRR is RAZ.</li> </ul> <p>For more information on these registers, see the <i>Arm® Cortex®-M52 Processor Technical Reference Manual</i>.</p>



- For more information on the ITCMCR and DTCMCR registers, see [ITCMCR and DTCMCR, TCM Control Registers](#).
- For more information on the ITGU\_CTRL and ITGU\_LUTn registers, see [TCM security gate registers](#).
- For more information on DTGU\_CTRL and DTGU\_LUTn registers, see [TCM security gate registers](#).
- For more information on the PAHBPCR register, see [PAHBPCR, P-AHB Control Register](#).
- For more information on the DCAICLR and DCAICRR registers, see [DCAICLR and DCADCLR, Direct Cache Access Location Registers](#) and [DCAICRR and DCADCRR, Direct Cache Access Read Registers](#).

## C.31 Error interface signals

The error interface reports *Error Correcting Code* (ECC) errors that are detected in the caches and TCMs. The processor can report the location of up to two errors which occur simultaneously. It can also indicate if more than two errors have occurred, but cannot provide any additional information. The following table shows the error interface signals.

**Table C-42: Error interface signals**

Signal name	Direction	Description
DMEV0	Output	This signal indicates that an error is detected. When this signal is asserted, DMEL0 and DMEI0[25:0] are valid.
DMEV1	Output	This signal indicates that at least two errors are detected. When this signal is asserted, DMEL1 and DMEI1[25:0] are valid.
DMEV2	Output	This signal indicates that at least three errors are detected. No information about errors beyond the first two is sent.
DMEL0[2:0]	Output	Location of the highest priority error detected. This is a one-hot signal and the format is:  <div> DMEL0[2]                      Error is found in the instruction cache.  DMEL0[1]                      Error found in the data cache.  DMEL0[0]                      Error found in the TCM. </div>
DMEL1[2:0]	Output	Location of the second highest priority error detected. This is a one-hot signal and the format is:  <div> DMEL1[2]                      Error is found in the instruction cache.  DMEL1[1]                      Error found in the data cache.  DMEL1[0]                      Error found in the TCM. </div>
DMEI0[25:0]	Output	Information about the highest priority error detected. This format of the signal depends on the location of the error:  <div> <b>Instruction cache</b>                      DMEI0[14:0] is the same format as bits [16:2] in IEBR0.  <b>Data cache</b>                              DMEI0[15:0] is the same format as bits [17:2] in DEBR0.  <b>TCM</b>    DMEI0[25:0] is the same format as bits [27:2] in TEBR0. </div> Unused bits of this signal are zero.
DMEI1[25:0]	Output	Information about the second highest priority error detected. This format of the signal depends on the location of the error:  <div> <b>Instruction cache</b>                      DMEI1[14:0] is the same format as bits [16:2] in IEBR1.  <b>Data cache</b>                              DMEI1[15:0] is the same format as bits [17:2] in DEBR1.  <b>TCM</b>    DMEI1[25:0] is the same format as bits [27:2] in TEBR1. </div> Unused bits of this signal are zero.
DBE[5:0]	Output	Detected Bus Error. A parity error has been detected from a protected interface.  <div> <b>Bit [5]</b>                      PMC-100 APB parity error  <b>Bit [4]</b>                      Debug AHB (D-AHB) parity error.  <b>Bit [3]</b>                      Main interface (M-AXI/M-AHB) parity error.  <b>Bit [2]</b>                      AHB TCM Access (TCM-AHB) parity error.  <b>Bit [1]</b>                      Peripheral AHB (P-AHB) parity error.  <b>Bit [0]</b>                      External Private Peripheral Bus (EPPB) parity error. </div> A single-cycle pulse on the associated bit of DBE signals an error. This signal is always 0b000000 if interface protection is not configured on the processor.

Signal name	Direction	Description
DFE[1:0]	Output	<p>Detected parity error from the flip-flop protection logic.</p> <p><b>1</b> IWIC flip-flop parity error <b>0</b> PDCORE flip-flop parity error</p> <p>An error in the IWIC is signaled on DFE[1] until nIWICRESET is asserted. An error in PDCORE is signaled on DFE[0] until nPORESET is asserted. If PDCORE enters retention, then DFE[0] is driven to 0 until PDCORE exits retention.</p> <p>This signal is always 0b00 if flop parity is not configured in the processor.</p> <p><b>Note:</b> There is an extra bit included in the DFE output signal from the MCU level which indicated a parity error detected in the EWIC.</p>

## C.32 Floating-point exception signals

The following table shows the floating-point exception signals.

The floating-point exception signals indicate mathematical errors that cause floating-point exceptions. Using these to indicate floating-point exceptions permits such exceptions to be diagnosed independently from software. For example, in safety-critical systems, exceptions can be routed directly to an on-chip safety controller.



Note

The floating-point exception signals are not related to the exception handling model. This means you can connect the floating-point exception signals to IRQ lines as your system design requires.

**Table C-43: Floating-point signals**

Signal name	Direction	Description
FPIX	Output	Masked floating-point inexact exception
FPIDC	Output	Masked floating-point input denormal exception
FPOFC	Output	Masked floating-point overflow exception
FPUFC	Output	Masked floating-point underflow exception
FPDZC	Output	Masked floating-point divide-by-zero exception
FPIOC	Output	Invalid operation

## C.33 PMC-100 interface signals

The following table shows the signals that are used only by the *Programmable MBIST Controller* (PMC-100). This interface contains control and configuration signals and PMC-100 APB signals

used by an external agent to program the PMC-100. If the PMC-100 is not included in your processor configuration, then these signals are not used and must be tied off.

**Table C-44: PMC-100 control and configuration signals**

Signal name	Direction	Description
PMCTEN	Input	<p>Test enable. This is the master hardware enable for PMC-100. When this signal is asserted, on-line MBIST transactions can occur. When this signal is deasserted and tied LOW:</p> <ul style="list-style-type: none"> <li>Only CoreSight™ registers and the internal PMC-100 control register are visible to reads from the memory mapped area in the PPB region. All other locations return zero.</li> <li>All writes to the memory-mapped area in the PPB region are ignored.</li> <li>On-line MBIST transactions do not occur.</li> </ul> <p><b>Note:</b> This signal is sampled only at reset.</p>
PMCTC	Input	Test continue pulse. This is a single cycle pulse and when enabled by an internal register in PMC-100, it causes a suspended test to continue execution.
PMCTE	Output	Test ended. When enabled in an internal register in PMC-100, this signal indicates that the test program has completed.
PMCTF	Output	<p>Test failed. When enabled in an internal register in the PMC-100, this signal indicates that a memory fault has been detected.</p> <p><b>Note:</b> PMCTF and PMCTE may be asserted at the same time.</p>

**Table C-45: PMC-100 APB interface signals**

Signal name	Direction	Description
PMCPSEL	Input	APB device select. Indicate that a data transfer is requested.
PMCPENABLE	Input	APB control signal. Strobe to time all accesses. Indicate the access phase of an APB transfer.
PMCPPROT[2:0]	Input	Transfer privilege and security level.
PMCPWRITE	Input	Write transfer.
PMCPSTRB[3:0]	Input	Write data byte strobes.
PMCPADDR[11:2]	Input	Transfer address.
PMCPWDATA[31:0]	Input	APB 32-bit write data bus.
PMCPREADY	Output	APB slave ready signal. This signal is driven LOW if the currently accessed APB device requires extra wait states to complete the transfer.
PMCPSLVERR	Output	APB slave error signal. This signal is driven HIGH if the currently accessed APB device cannot handle the requested transfer.
PMCPRDATA[31:0]	Output	APB 32-bit read data bus.

### C.33.1 PMC-100 APB interface protection signals

The following table shows the PMC-100 APB interface protection signals.

**Table C-46: PMC-100 APB interface protection signals**

Signal name	Direction	Description
PMCPSELCHK	Input	Odd parity of PMCPSEL.
PMCPREADYCHK	Output	Odd parity of PMCPREADY.
PMCPMCPENABLECHK	Input	Odd parity of PMCPENABLE.
PADDRCHK[1:0]	Input	Odd parity of {PADDR[11:8], [7:2]}.
PMCPRDATACHK[3:0]	Output	Odd parity of PMCPRDATA[31:0] at 8-bit granularity.
PMCPWDATACHK[3:0]	Input	Odd parity of PMCPWDATA[31:0] at 8-bit granularity.
PMCPCTRLCHK	Input	Odd parity of PMCPROT[2:0], PWRITE.
PMCPSTRBCHK	Input	Odd parity of PMCPSTRB[3:0].
PMCPSLVERRCHK	Output	Odd parity of PMCPSLVERR.

## C.34 Test interface signals

The following tables show the *Design for Test* and production *Memory Built-In Self-Test* (MBIST) interface signals.

**Table C-47: DFT signals**

Signal name	Direction	Description
DFTCGEN	Input	Enables architectural clock gate override.
DFTRSTDISABLE[1:0]	Input	Disables synchronized multi-layer logic resets during scan shift.  <b>DFTRSTDISABLE[0]</b> Disables the first level reset logic. <b>DFTRSTDISABLE[1]</b> Disables the second level reset logic.
DFTRAMHOLD	Input	Disable writes to the RAMs during scan shift.

**Table C-48: Production MBIST interface signals**

Signal name	Direction	Description
MBISTREQ	Input	Production MBIST mode request.  <b>0</b> Normal operation <b>1</b> Production MBIST mode



## C.35 DCLS operation signals

The following table shows the signals that the processor uses when configured with *Dual-Core Lock-Step* (DCLS) operation.

**Table C-49: DCLS operation signals**

Signal name	Direction	Description
CLKINDCLS	Input	Primary clock for the redundant processor logic. This signal must be synchronous to CLKIN.
IWICCLKDCLS	Input	Clock for the redundant IWIC logic. This clock must be synchronous to IWICCLK
nPORESETDCLS	Input	Cold reset for the redundant processor logic. This signal must be asserted and deasserted together with nPORESET. If CLKIN is active when nPORESETDCLS is asserted or deasserted, then the signal must be constrained such that nPORESETDCLS is stable on the rising edge of the clock.
nSYSRESETDCLS	Input	Warm reset for the redundant processor logic. This signal must be asserted and deasserted together with nSYSRESET. If CLKIN is active when nSYSRESETDCLS is asserted or deasserted, then the signal must be constrained such that nSYSRESETDCLS is stable on the rising edge of the clock
nIWICRESETDCLS	Input	Cold reset for the redundant IWIC logic. This signal must be asserted and deasserted together with nIWICRESET. If IWICCLK is active when nIWICRESETDCLS is asserted or deasserted, then the signal must be constrained such that nIWICRESETDCLS is stable on the rising edge of the clock
DCLSCORECTL[11:0]	Input	Core DCLS feature control. For more information on bit assignments, see <a href="#">Control and reporting</a> .
DCLSIWICCTL[5:0]	Input	IWIC DCLS feature control. For more information on bit assignments, see <a href="#">Control and reporting</a> .
DCLSCORECOMPRES[11:0]	Output	Core DCLS comparator results. For more information on bit assignments, see <a href="#">Control and reporting</a> .
DCLSIWICCOMPRES[5:0]	Output	IWIC DCLS comparator results. For more information on bit assignments, see <a href="#">Control and reporting</a> .

### C.35.1 Control and reporting

DCLS operation is controlled using the DCLSCORECTL and DCLSIWICCTL input signals, while reporting comparator matches and other errors are handled through the DCLSCORECOMPRES and DCLSIWICCOMPRES output signals.

These signals are divided into nine fields. Each field is 2 bits wide. The first six fields refer to defined areas of processor activity. The remaining three fields are optionally available for IWIC activity.

For each field, the two DCLSCORECTL and DCLSIWICCTL input bits control comparator use. The two DCLSCORECOMPRES and DCLSIWICCOMPRES bits report the primary and secondary comparator results for that area.

The core and RAM signals have the following fields:

**Table C-50: Core and RAM fields**

Field name	Description	Notes
IA_ACTV	Inadvertent Activation Zone Activity	<p>These bits control and report activity in areas of the design which are not protected by duplicate logic but which directly affect the safety-critical logic in the design.</p> <p>Some logic can come from areas of the Cortex®-M52-based system that are not protected by the <i>Dual-Core Lock-Step</i> (DCLS) functionality. This logic is not intended to be used in safety critical applications, however, if this logic can cause a change in the behavior of safety critical application-protected logic, then it must be indicated as being active to the system on the DCLSCOMPRES and DCLSIWICCOMPRES output signals.</p> <p>This is a requirement because the input might be tied to a particular value as a result of expected behavior or it could be tied to a particular value as a result of a fault in the logic that is not protected by the DCLS functionality, and Cortex®-M52 cannot distinguish between these cases. The system must be aware of the possibility that the fault might have been propagated without being identified.</p>
IA_ON	Inadvertent Activation Zone Enabled	These bits control and report that non-safety protected logic, that is, logic that is not duplicated, is active (but not necessarily directly affecting the safety critical logic in the design).
Core	Processor core	These bits provide control and reporting of safety functionality for the duplicated processor logic that is located in the PDCORE domain.
RAM	Processor embedded RAM zone	These bits provide control and reporting of safety functionality for the RAM interfaces
Core Reset	Reset functionality	These bits provide control and reporting of safety functionality for the reset signals into the processor. A primary and redundant reset is required from the system.
Core Clock	Clocking	These bits provide control and reporting of safety functionality for the clock signals into the processor. A primary and redundant clock is required from the system.

The IWIC signals have the following fields:

**Table C-51: IWIC fields**

Field name	Description
IWIC zone	Represents the monitors for the IWIC.
IWIC Reset	Monitors the reset signal into the IWIC. It requires a second independent IWIC reset to be supplied to the design.
IWIC Clock	Monitors the clock signal into the core. It requires a second independent IWIC clock to be supplied to the processor.

The following table shows the meaning of encoded value for each field of the DCLSCORECTL and DCLSIWICCTL signal.

**Table C-52: Meaning of fields value in the DCLSCORECTL and DCLSIWICCTL signal**

Value	Action
00	Field disable
11	Field enable
10	Force fault to field
01	Force fault to field

The following table shows the bit assignments for the DCLSCORECOMPRES signal.

In the following table:

**PRIM**

Primary logic comparator has detected a difference between the primary logic and the secondary logic output.

**SEC**

Secondary logic comparator has detected a difference between the primary logic and the secondary logic output.

**Table C-53: DCLSCORECOMPRES signals**

Bits	Field name	Results
[11]	IA_ACTV	SEC
[10]		PRIM
[9]	IA_ON	SEC
[8]		PRIM
[7]	Core	SEC
[6]		PRIM
[5]	RAM	SEC
[4]		PRIM
[3]	Reset	SEC
[2]		PRIM
[1]	Clock	SEC
[0]		PRIM

The following table shows the bit assignments for the DCLSIWICCOMPRES signal.

In the following table:

**PRIM**

Primary logic comparator has detected a difference between the primary logic and the secondary logic output.

**SEC**

Secondary logic comparator has detected a difference between the primary logic and the secondary logic output.

**Table C-54: DCLSIWICCOMPRES signals**

Bits	Field name	Results
[5]	IWIC	SEC
[4]		PRIM
[3]	Reset	SEC
[2]		PRIM
[1]	Clock	SEC
[0]		PRIM

# Appendix D UNPREDICTABLE Behaviors

This appendix summarizes the behavior of the Cortex®-M52 processor in cases where the Arm®v8.1-M architecture is **UNPREDICTABLE**.

## D.1 Use of instructions defined in architecture variants

An instruction that is provided by one or more of the architecture extensions is either **UNPREDICTABLE** or **UNDEFINED** in an implementation that does not include those extensions.

In the Cortex®-M52 processor, all instructions that are not explicitly supported generate an UNDEFINSTR UsageFault exception.

## D.2 Use of Program Counter - R15 encoding

R15 is **UNPREDICTABLE** as a source or destination in most data processing operations. R15 is also **UNPREDICTABLE** as a transfer register in certain load/store instructions. Examples of such instructions include LDRT, LDRH, and LDRB.

In the Cortex®-M52 processor, the use of R15 as a named register specifier for any source or destination register that is indicated as **UNPREDICTABLE** generates an UNDEFINSTR UsageFault exception.

## D.3 Use of Stack Pointer - as a general-purpose register R13

R13 is defined in the Thumb instruction set so that its use is primarily as a stack pointer. R13 is normally identified as stack pointer, SP in Thumb instructions.

In 32-bit Thumb instructions, if you use SP as a general-purpose register beyond the architecturally defined constraints, the results are **UNPREDICTABLE**.

In the Cortex®-M52 processor, the use of R13 as a named register specifier for any source or destination register that is indicated as **UNPREDICTABLE** generates an UNDEFINSTR UsageFault exception.

In the architecture where the use of R13 as a general-purpose register is defined, bits[1:0] of the register must be treated as SBZP. Writing a nonzero value to bits [1:0] results in **UNPREDICTABLE** behavior. In the Cortex®-M52 processor, bits [1:0] of R13 are always RAZ/WI.

## D.4 Register list in load and store multiple instructions

Load and Store Multiple instructions (`LDM`, `STM`, `PUSH`, `POP`, `VLDM`, and `VSTM`) transfer multiple registers to and from consecutive memory locations using an address from a base register, which can be optionally written back when the operation is complete.

The registers are selected from a list encoded in the instruction. Some of these encodings are **UNPREDICTABLE**.

In the Cortex®-M52 processor:

- If the number of registers loaded is zero, then the instruction is a *No Operation* (NOP). If the number of registers loaded is one, the single register is loaded.
- For a Load Multiple, if PC is specified in the list and the instruction is in an IT block, a fault is not generated. If the branch is taken, then the IT state is cleared.
- For a Store Multiple instruction, if PC is specified in the list, an **UNDEFINED** exception occurs.
- For a Load Multiple instruction, if base writeback is specified and the register to be written back is also in the list to be loaded, the instruction performs all the loads in the specified addressing mode and the register being written back takes the loaded value.
- For a Store Multiple instruction, if base writeback is specified and the register to be written back is also the first register in the list to be stored, the value stored is the initial base register value. The base register is written back with the expected updated value. If the register to be written back is not the first register in the list, then it takes the updated value.
- For a floating-point Load or Store Multiple instruction, `VLDM`, `VSTM`, `VPUSH`, and `VPOP`, if the register list extends beyond S63 or D31, then the Cortex®-M52 processor ignores all registers that are greater than S31 or D15. If it has base writeback, then the base register becomes **UNKNOWN**.

## D.5 Exception-continuable instructions

To improve interrupt response and increase processing throughput, the processor can take an interrupt during the execution of a Load Multiple or Store Multiple instruction, and continue execution of the instruction after returning from the interrupt. During the interrupt processing, the EPSR.ICI bit holds the continuation state of the Load Multiple or Store Multiple instruction.

In the Cortex®-M52 processor, if an exception-continuable instruction is interrupted, then modification of the EPSR.ICI bits by either the software or a debugger might generate an INVSTATE UsageFault exception when re-execution of the interrupted instruction is attempted.

This includes the architecturally **UNPREDICTABLE** cases of:

- Not a register in the register list of the Load Multiple or Store Multiple instruction.
- The first register in the register list of the Load Multiple or Store Multiple instruction.

The Cortex®-M52 processor also generates an INVSTATE UsageFault exception if the ICI bits are set to any non-zero value for an integer Load Multiple instruction with the base register in the

register list, and ICI set to a greater register number than the base register. This is because these instructions are not eligible for continuation.

## D.6 Stack limit checking

The Arm®v8.1-M architecture defines the instructions which are subject to stack limit checking when operating on SP.

It states that it is **UNKNOWN** whether a stack limit check is performed on any use of the SP that was **UNPREDICTABLE** in Arm®v7-M and Arm®v6-M.

In the Cortex®-M52 processor, stack limit checking behaves as follows:

- Instructions write back to SP and are not defined as **UNPREDICTABLE** are checked.
- Exception entry and Tail-chaining are checked.
- Load instructions with the SP as the destination are not checked.
- The result of stack limit checking becomes **UNKNOWN** for the cases of stack overflow and underflow.

## D.7 UNPREDICTABLE instructions within an IT block

Instructions executed in an IT block which change the PC are architecturally **UNPREDICTABLE** unless they are the last instruction in the block.

In an IT block, the behaviors of Cortex®-M52 processor are as follows:

- Conditional branch instructions (**Bcond label**) always generate an UNCONDITIONAL UNDEFINSTR UsageFault exception.
- Unconditional branch instructions (**B label**) execute normally anywhere in the IT block.
- Branch with link instructions (**BL label**) execute normally anywhere in the IT block.
- **BLX** PC is always **UNPREDICTABLE** and generates an UNDEFINSTR UsageFault exception.
- Branch and exchange instructions (**BX Rm**) execute normally anywhere in the IT block.
- Compare and Branch instruction, **CBNZ** and **CBZ** always generate an UNCONDITIONAL UNDEFINSTR UsageFault exception.
- Table branch instructions (**TBB** and **TBH**) execute normally anywhere in the IT block.
- An **IT** instruction inside another IT block always generates an UNCONDITIONAL UNDEFINSTR UsageFault exception.
- If the Floating-point Extension is included and one of the following instructions is executed in an IT block, the instruction generates an unconditional UNCONDITIONAL UNDEFINSTR UsageFault exception:
  - **VCVTA**

- VCVTN
  - VCVTP
  - VCVTM
  - VMAXNM
  - VMINNM
  - VRINTA
  - VRINTN
  - VRINTP
  - VRINTM
  - VSEL
- cps instructions always generate an UNCONDITIONAL UNDEFINSTR UsageFault exception.
  - cx1, cx2, and cx3 are **UNPREDICTABLE** in IT blocks and are always unconditionally UNDEFINED.
  - vcx1, vcx2, vcx3, vcx1A, vcx2A, and vcx3A are always **UNPREDICTABLE** in IT blocks and always unconditionally UNDEFINED.

## D.8 Memory access and address space

In the Arm®v8.1-M architecture, there are memory accesses that result in **UNPREDICTABLE** behavior in the Cortex®-M52 processor.

The following table shows the memory accesses that are **UNPREDICTABLE** and the Cortex®-M52 processor behavior.

**Table D-1: Memory accesses and Cortex®-M52 processor behavior**

Memory access	Cortex®-M52 processor behavior
Any access to memory from a load or store instruction or an instruction fetch, which overflows the 32-bit address space.	These kinds of accesses wrap around to addresses at the start of memory.
For any access X, the bytes accessed by X must all have the same memory type attribute, otherwise the behavior of the access is <b>UNPREDICTABLE</b> . That is, an unaligned access that spans a boundary between different memory types is <b>UNPREDICTABLE</b> .	In the Cortex®-M52 processor, each part of an access to a different 32-byte aligned region is dealt with independently. If an MPU is included in the processor, each access to a different 32-byte region makes a new MPU lookup. If an MPU is not included, then the behavior of the associated background region is taken into account.
For any two memory accesses X and Y that are generated by the same instruction, the bytes accessed by X and Y must all have the same memory type attribute. Otherwise, the results are <b>UNPREDICTABLE</b> . For example, an LDC, LDM, LDRD, STC, STM, STRD, VSTM, VLDM, VPUSH, VPOP, VLDR, or VSTR that spans a boundary between Normal and Device memory is <b>UNPREDICTABLE</b> .	In the Cortex®-M52 processor, each part of access to a different 32-byte aligned region is dealt with independently. If an MPU is included in the processor, each access to a different 32-byte aligned region makes a new MPU lookup. If an MPU is not included, then the behavior of the associated background region is taken into account.
Any instruction fetch must only access Normal memory. If it accesses Device memory, the result is <b>UNPREDICTABLE</b> . For example, instruction fetches must not be performed to an area of memory that contains read-sensitive devices because there is no ordering requirement between instruction fetches and explicit accesses.	In the Cortex®-M52 processor, fetches to Device memory are sent out to the system, indicated on the Main interface as Device, unless the memory region is marked with the <i>Execute Never</i> (XN) memory attribute.



Memory access	Cortex®-M52 processor behavior
If the Security Extension is implemented, the behavior of sequential instruction fetches that cross from Non-secure to Secure memory and fulfill the secure entry criteria specified in the architecture, including the presence of a <i>Secure Gateway</i> (SG) instruction at the boundary of the secure memory area, is <b>CONSTRAINED UNPREDICTABLE</b> .	In the Cortex®-M52 processor, this results in a fault (INVEP).

## D.9 MPU programming

The Arm® *Protected Memory System Architecture* (PMSA) includes many **UNPREDICTABLE** cases when programming the MPU when it is included in an implementation.

In the Cortex®-M52 processor:

- Setting MPU\_CTRL.ENABLE to 0 and MPU\_CTRL.HFNMIENA to 1 is **UNPREDICTABLE**. This results in all memory accesses using the default memory map including those from Exception Handlers with a priority less than one.
- If MPU\_RNR is written with a region number greater than the number of regions defined in the MPU, then the value used is masked by one less than the number of regions defined. For example:
  - The number of regions defined is given as num\_regions. The value written to MPU\_RNR is given as v.
  - num\_regions=8 and v=9.
  - The effective region used is given as 9 & (8-1); region 1.
 The number of regions available can be read from MPU\_TYPE.DREGION.
- Setting MPU\_RBAR.SH to 1 is **UNPREDICTABLE**. This encoding is treated as Non-shareable.
- The Attribute fields (MPU\_ATTR) of the MPU\_MAIR0 and MPU\_MAIR1 registers include some encodings which are **UNPREDICTABLE**.
  - If MPU\_ATTR[7:4]!=0 and MPU\_ATTR[3:0]==0 is **UNPREDICTABLE**, the attributes are treated as Normal memory, Outer non-cacheable, Inner non-cacheable.
  - If MPU\_ATTR[7:4]==0 and MPU\_ATTR[1:0]!=0 is **UNPREDICTABLE**, the attributes are treated as Device-nGnRE.
- The external AMBA® 5 AHB interface signals cannot distinguish between some of the memory attribute encodings defined by the PMSA:
  - Normal transient memory is treated the same as Normal non-transient memory.
  - Device memory with Gathering or Reordering attributes (G, R) are always treated as non-Gathering and non-Reordering. Early Write Acknowledgment attributes (E, nE) are supported on the Cortex®-M52 AHB5 interfaces.

## D.10 Miscellaneous UNPREDICTABLE instruction behavior

This section documents the behavior of the Cortex®-M52 processor in a number of miscellaneous **UNPREDICTABLE** instruction scenarios:

- Load instructions that specify writeback of the base register are **UNPREDICTABLE** if the base register to be written back matches the register to be loaded ( $Rn==Rt$ ). These cases execute normally and overwrite base register with load back value.
- Store instructions that specify writeback of the base register are **UNPREDICTABLE** if the base register to be written back matches the register to be stored ( $Rn==Rt$ ). In the Cortex®-M52 processor, the value stored is the initial base register value. The base register is then written back with the expected updated value. These cases generate an **UNDEFINSTR** UsageFault exception.
- Multiply and multiply accumulate instructions that write a 64-bit result using two registers, **SMULL**, **SMLAL**, **SMLALBB**, **SMLALBT**, **SMLALTB**, **SMLALTT**, **SMLALD**, **SMLALDX**, **SMLSLD**, **SMLSLDX**, **UMULL**, and **UMAAL** are **UNPREDICTABLE** if the two registers are the same ( $RdHi==RdLo$ ). In the Cortex®-M52 processor, these cases generate an **UNDEFINSTR** UsageFault exception.
- Floating-point instructions that transfer between two registers and either two single-precision registers or one double-precision register, **VMOV Rt, Rt2, Dm** and **VMOV Rt, Rt2, Sm, Sm1** are **UNPREDICTABLE** if the two registers are the same ( $Rt==Rt2$ ). In the Cortex®-M52 processor, these cases generate an **UNDEFINSTR** UsageFault exception.

# Appendix E Revisions

This appendix describes the technical changes between released issues of this book.

## E.1 Revisions

The following table shows the significant technical changes of this book.

**Table E-1: Issue 0000-01**

Change	Location
First Beta release for r0p0	-

**Table E-2: Differences between issue 0000-01 and 0000-02**

Change	Location
First limited access release for r0p0	-
Removed the information of data prefetch	Cortex-M52 processor core
Added the following implementation options: <ul style="list-style-type: none"> <li>Flop parity protection</li> <li>SBIST relevant observation register</li> </ul>	Cortex-M52 implementation options
Added information about the unified cache.	System registers
Updated the information about direct cache access registers	Direct cache access registers
The STLD0MPUOR observation register is changed to STLD0MPUOR, and the STLD1MPUOR register is removed.	STLIDMPUSR, STLIMPUOR, and STLDMPUOR, MPU observation registers  IMPLEMENTATION DEFINED registers summary
Updated the information about MSCR	MSCR, Memory System Control Register
Updated the information about processor configuration information registers	Processor configuration information registers
Removed information about the LOGIC_RET (Cache) power mode.	Power states  Q-Channel clock control
Updated the following information: <ul style="list-style-type: none"> <li>High performance configuration M-AXI attributes and transaction</li> <li>TCM interfaces</li> <li>Direct cache access</li> </ul>	High performance configuration M-AXI attributes and transactions  TCM interfaces  Direct cache access
For ECC memory protection behavior, added information about the unified cache.	ECC memory protection behavior
Updated the following information: <ul style="list-style-type: none"> <li>Processor ROM table identification and entries</li> <li>Debug identification block register summary</li> </ul>	Processor ROM table identification and entries  Debug identification block register summary
Updated the following PMU information: <ul style="list-style-type: none"> <li>The table of PMU events</li> <li>The reset value of some PMU registers</li> </ul>	Performance Monitoring Unit Extension

Change	Location
<p>Updated the following register summary:</p> <ul style="list-style-type: none"> <li>ITM register summary</li> <li>DWT register summary</li> <li>CTI register summary</li> <li>BPU register summary</li> <li>EWIC CoreSight™ register summary</li> </ul>	<p><a href="#">ITM register summary</a></p> <p><a href="#">DWT register summary</a></p> <p><a href="#">CTI register summary</a></p> <p><a href="#">BPU register summary</a></p> <p><a href="#">EWIC CoreSight register summary</a></p>
<p>The following signals are updated:</p> <ul style="list-style-type: none"> <li>Reset signals</li> <li>Static configuration signals</li> <li>Cache initialization signal</li> <li>TCM-AHB interface signals</li> <li>IDAU interface signals</li> <li>PMC-100 interface signals</li> <li>DCLS control and reporting</li> </ul>	<p><a href="#">Signal Descriptions</a></p>
<p>Under UNPREDICTABLE Behaviors, the following information is updated:</p> <ul style="list-style-type: none"> <li>Stack limit checking</li> <li>UNPREDICTABLE instructions within an IT block</li> </ul>	<p><a href="#">UNPREDICTABLE Behaviors</a></p>

**Table E-3: Differences between issue 0000-02 and 0001-03**

Change	Location
First early access release for r0p1	-
Cortex®-M52 implementation options are updated.	<a href="#">Cortex-M52 implementation options</a>
<p>In the Programmers model chapter, the following topics are modified:</p> <ul style="list-style-type: none"> <li>Security states, operation, and execution modes</li> <li>Exceptions</li> </ul>	<a href="#">Programmers model</a>
<p>In the System register chapter, minor changes are made to the following topics:</p> <ul style="list-style-type: none"> <li>System control register summary</li> <li>Media and VFP Feature Register reset values, MVFR0, MVFR1, and MVFR2 reset values</li> <li>IMPLEMENTATION DEFINED registers summary</li> <li>Power mode control registers</li> <li>CPDLPSTATE, Core Power Domain Low Power State Register</li> </ul>	<a href="#">System registers</a>
<p>In the Power management chapter, minor changes are made to the following topics:</p> <ul style="list-style-type: none"> <li>Core P-Channel and power mode selection</li> <li>COREPACTIVE and required power mode</li> </ul>	<a href="#">Power management</a>
In the Memory Authentication chapter, the Security check and fault response topic is updated.	<a href="#">Security check and fault response</a>

Change	Location
In the Memory system chapter, minor changes are made to the following topics: <ul style="list-style-type: none"> <li>Memory system features</li> <li>Memory system faults and the classes of fault</li> <li>Speculative accesses</li> <li>High performance M-AXI configuration</li> <li>Restrictions on AXI transfers</li> <li>AHB Main (M-AHB) interface</li> <li>TCM interfaces</li> <li>Instruction cache, data cache, and unified cache</li> <li>Store buffer</li> </ul>	Memory system
In the Reliability, Availability, and Serviceability Extension support chapter, the following topics are updated: <ul style="list-style-type: none"> <li>RAS events</li> <li>Interface protection behavior</li> </ul>	Cortex-M52 RAS events Interface protection behavior
Floating-point and MVE operation section updated	Floating-point and MVE operation
The following signal topics are updated: <ul style="list-style-type: none"> <li>Reset signals</li> <li>Data Tightly Coupled Memory interface signals</li> <li>TCM-AHB interface protection signals</li> <li>P-AHB interface signals</li> <li>P-AHB interface protection signals</li> <li>PMC-100 interface signals</li> <li>Control and reporting</li> </ul>	Reset signals Data Tightly Coupled Memory interface signals TCM-AHB interface protection signals P-AHB interface signals P-AHB interface protection signals PMC-100 interface signals Control and reporting

**Table E-4: Differences between issue 0001-03 and 0001-04**

Change	Location
Second early access release for r0p1	-
Minor changes to fix editing errors	The whole document

**Table E-5: Differences between issue 0001-04 and 0002-05**

Change	Location
First release for r0p2	-
Updated the revision number to r0p2	-

**Table E-6: Differences between issue 0002-05 and 0002-06**

Change	Location
Second release for r0p2	-
Change the product name from Mizar to Cortex®-M52	-