

# ARM® Cortex®-R7 MPCore

Revision: r0p1

## Technical Reference Manual



# ARM Cortex-R7 MPCore

## Technical Reference Manual

Copyright © 2012, 2014 ARM. All rights reserved.

### Release Information

The following changes have been made to this book.

Change history			
Date	Issue	Confidentiality	Change
21 March 2012	A	Non-Confidential	First release for r0p0
28 September 2012	B	Non-Confidential	First release for r0p1
28 November 2014	C	Non-Confidential	Second release for r0p1

### 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. 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 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 makes no representation with respect to, and has undertaken no analysis to identify or understand the scope and content of, third party 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 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 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’s customers is not intended to create or refer to any partnership relationship with any other company. ARM may make changes to this document at any time and without notice.

If any of the provisions contained in these terms conflict with any of the provisions of any signed written agreement covering this document with ARM, then the signed written agreement prevails over and supersedes the conflicting provisions of these terms. 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.

Words and logos marked with ® or ™ are registered trademarks or trademarks of ARM Limited or its affiliates in the EU and/or elsewhere. All rights reserved. Other brands and names mentioned in this document may be the trademarks of their respective owners. Please follow ARM’s trademark usage guidelines at <http://www.arm.com/about/trademark-usage-guidelines.php>

Copyright ©, ARM Limited or its affiliates. All rights reserved.

ARM Limited. Company 02557590 registered in England.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

**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 and the party that ARM delivered this document to.

**Product Status**

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

**Web Address**

<http://www.arm.com>

# Contents

## ARM Cortex-R7 MPCore Technical Reference Manual

	<b>Preface</b>	
	About this book .....	viii
	Feedback .....	xii
<b>Chapter 1</b>	<b>Introduction</b>	
	1.1 About the Cortex-R7 MPCore processor .....	1-2
	1.2 Compliance .....	1-3
	1.3 Features .....	1-4
	1.4 Interfaces .....	1-5
	1.5 Configurable options .....	1-6
	1.6 Redundant processor comparison .....	1-8
	1.7 Test features .....	1-9
	1.8 Product documentation and design flow .....	1-10
	1.9 Product revisions .....	1-12
<b>Chapter 2</b>	<b>Functional Description</b>	
	2.1 About the functions .....	2-2
	2.2 Interfaces .....	2-4
	2.3 Clocking, resets, and initialization .....	2-5
	2.4 Power management .....	2-13
	2.5 Processor ports .....	2-19
<b>Chapter 3</b>	<b>Programmers Model</b>	
	3.1 About the programmers model .....	3-2
	3.2 The VFP extension .....	3-3
	3.3 Multiprocessing extensions .....	3-4

	3.4	Memory formats .....	3-5
	3.5	Addresses in the Cortex-R7 MPCore processor .....	3-6
<b>Chapter 4</b>		<b>System Control</b>	
	4.1	About system control .....	4-2
	4.2	Register summary .....	4-3
	4.3	Register descriptions .....	4-17
<b>Chapter 5</b>		<b>Floating Point Unit Programmers Model</b>	
	5.1	About the FPU programmers model .....	5-2
	5.2	IEEE 754 standard compliance .....	5-3
	5.3	Instruction throughput and latency .....	5-4
	5.4	Register summary .....	5-6
	5.5	Register descriptions .....	5-8
<b>Chapter 6</b>		<b>Level One Memory System</b>	
	6.1	About the L1 memory system .....	6-2
	6.2	Fault handling .....	6-3
	6.3	About the TCMs .....	6-7
	6.4	About the caches .....	6-8
	6.5	Local exclusive monitor .....	6-17
	6.6	Memory types and L1 memory system behavior .....	6-18
	6.7	Error detection events .....	6-19
<b>Chapter 7</b>		<b>Fault Detection</b>	
	7.1	About fault detection .....	7-2
	7.2	RAM protection .....	7-3
	7.3	Logic protection .....	7-10
	7.4	External memory and bus protection .....	7-11
	7.5	Programmers view .....	7-12
	7.6	Lock-step .....	7-14
	7.7	Static split/lock .....	7-16
<b>Chapter 8</b>		<b>Determinism Support</b>	
	8.1	About determinism support .....	8-2
	8.2	Memory Protection Unit .....	8-3
	8.3	Branch prediction .....	8-11
	8.4	Low latency interrupt mode .....	8-12
	8.5	System configurability and QoS .....	8-13
	8.6	Instruction and data TCM .....	8-15
<b>Chapter 9</b>		<b>Multiprocessing</b>	
	9.1	About multiprocessing and the SCU .....	9-2
	9.2	Multiprocessing programmers view .....	9-4
	9.3	SCU registers .....	9-5
	9.4	Interrupt controller .....	9-19
	9.5	Private timer and watchdog .....	9-29
	9.6	Global timer .....	9-35
	9.7	Accelerator Coherency Port .....	9-39
<b>Chapter 10</b>		<b>Monitoring, Trace, and Debug</b>	
	10.1	About monitoring, trace, and debug .....	10-2
	10.2	Performance Monitoring Unit .....	10-3
	10.3	Memory Reconstruction Port .....	10-10
	10.4	Embedded Trace Macrocell .....	10-11
	10.5	Debug .....	10-12

**Chapter 11****Level Two Interface**

11.1	About the L2 interface .....	11-2
11.2	Optimized accesses to the L2 memory interface .....	11-8
11.3	Accessing RAMs using the AXI3 interface .....	11-9
11.4	STRT instructions .....	11-10
11.5	Event communication with an external agent using WFE/SEV .....	11-11
11.6	Accelerator Coherency Port interface .....	11-12

**Appendix A****Signal Descriptions**

A.1	About the signal descriptions .....	A-2
A.2	Clock and control signals .....	A-3
A.3	Reset signals .....	A-5
A.4	Interrupt controller signals .....	A-6
A.5	Configuration signals .....	A-7
A.6	Standby signals .....	A-9
A.7	Power management signals .....	A-10
A.8	AXI3 interfaces .....	A-11
A.9	Performance monitoring signals .....	A-23
A.10	Exception flag signals .....	A-24
A.11	Error detection notification signals .....	A-25
A.12	Test interface .....	A-31
A.13	MBIST interface .....	A-32
A.14	External debug signals .....	A-34
A.15	ETM/ATB interface signals .....	A-37
A.16	Memory reconstruction port signals .....	A-39
A.17	Power gating interface signals .....	A-40

**Appendix B****Cycle Timings and Interlock Behavior**

B.1	About instruction cycle timing .....	B-2
B.2	Data-processing instructions .....	B-3
B.3	Load and store instructions .....	B-4
B.4	Multiplication instructions .....	B-7
B.5	Branch instructions .....	B-8
B.6	Serializing instructions .....	B-9

**Appendix C****Revisions**

# Preface

This preface introduces the *ARM® Cortex®-R7 MPCore Technical Reference Manual*. It contains the following sections:

- [About this book on page viii.](#)
- [Feedback on page xii.](#)

## About this book

This book is for the Cortex-R7 MPCore processor.

## Product revision status

The *rn**pn* identifier indicates the revision status of the product described in this book, where:

- rn** Identifies the major revision of the product.
- pn** Identifies the minor revision or modification status of the product.

## Intended audience

This book is written for system designers, system integrators, and programmers who are designing or programming a *System-on-Chip* (SoC) that uses the Cortex-R7 MPCore processor.

## Using this book

This book is organized into the following chapters:

### Chapter 1 *Introduction*

Read this for an introduction to the Cortex-R7 MPCore processor.

### Chapter 2 *Functional Description*

Read this for descriptions of the major functional blocks.

### Chapter 3 *Programmers Model*

Read this for a description of the Cortex-R7 MPCore processor registers and programming information.

### Chapter 4 *System Control*

Read this for a description of the system control coprocessor registers and programming information.

### Chapter 5 *Floating Point Unit Programmers Model*

Read this for a description of the *Floating Point Unit* (FPU) support.

### Chapter 6 *Level One Memory System*

Read this for a description of the *Level One* (L1) memory system.

### Chapter 7 *Fault Detection*

Read this for a description of the fault detection features, including *Error Correcting Code* (ECC), lock-step, and split/lock.

### Chapter 8 *Determinism Support*

Read this for a description of the determinism support features, including the *Memory Protection Unit* (MPU) and *Quality of Service* (QoS).

### Chapter 9 *Multiprocessing*

Read this for a description of the multiprocessing features, including the *Snoop Control Unit* (SCU), interrupt controller, timers and watchdog, and *Accelerator Coherency Port* (ACP).

### Chapter 10 *Monitoring, Trace, and Debug*

Read this for a description of the monitoring, trace, and debug features, and the Integration Test Registers.



## Chapter 11 *Level Two Interface*

Read this for a description of the *Level Two* (L2) interface.

## Appendix A *Signal Descriptions*

Read this for a description of the Cortex-R7 MPCore processor signals.

## Appendix B *Cycle Timings and Interlock Behavior*

Read this for a description of the Cortex-R7 MPCore instruction cycle timing.

## Appendix C *Revisions*

Read this for a description of the technical changes between released issues of this book.

## 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 *ARM® Glossary* <http://infocenter.arm.com/help/topic/com.arm.doc.aeg0014-/index.html>.

## Conventions

Conventions that this book can use are described in:

- *Typographical conventions.*
- *Timing diagrams on page x.*
- *Signals on page x.*

### Typographical conventions

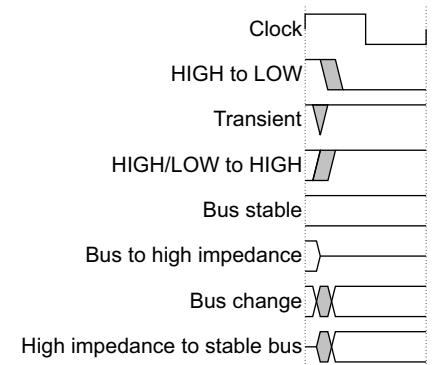
The following table describes the typographical conventions:

Style	Purpose
<i>italic</i>	Introduces special terminology, denotes cross-references, and citations.
<b>bold</b>	Highlights interface elements, such as menu names. Denotes signal names. Also used for terms in descriptive lists, where appropriate.
monospace	Denotes text that you can enter at the keyboard, such as commands, file and program names, and source code.
<u>monospace</u>	Denotes a permitted abbreviation for a command or option. You can enter the underlined text instead of the full command or option name.
monospace <i>italic</i>	Denotes arguments to monospace text where the argument is to be replaced by a specific value.
<b>monospace bold</b>	Denotes language keywords when used outside example code.
<and>	Enclose replaceable terms for assembler syntax where they appear in code or code fragments. For example: MRC p15, 0 <Rd>, <CRn>, <CRm>, <Opcode_2>
SMALL CAPITALS	Used in body text for a few terms that have specific technical meanings, that are defined in the <i>ARM Glossary</i> . For example, IMPLEMENTATION DEFINED, IMPLEMENTATION SPECIFIC, UNKNOWN, and UNPREDICTABLE.

## Timing diagrams

The figure named *Key to timing diagram conventions* 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.



**Key to timing diagram conventions**

## Signals

The signal conventions are:

- |                     |  |
|---------------------|--|
| <b>Signal level</b> | The level of an asserted signal depends on whether the signal is active-HIGH or active-LOW. Asserted means: <ul style="list-style-type: none"> <li>• HIGH for active-HIGH signals.</li> <li>• LOW for active-LOW signals.</li> </ul> |
| <b>Lower-case n</b> | At the start or end of a signal name denotes an active-LOW signal.   |

## Additional reading

This section lists publications by ARM and by third parties.

See Infocenter <http://infocenter.arm.com>, for access to ARM documentation.

## ARM publications

This book contains information that is specific to this product. See the following documents for other relevant information:

- *ARM® Architecture Reference Manual ARMv7-A and ARMv7-R edition* (ARM DDI 0406).
- *ARM Embedded Trace Macrocell Architecture Specification, ETMv4* (ARM DDI 0406).
- *ARM® CoreSight™ ETM-R7 Technical Reference Manual* (ARM DDI 0459).
- *ARM® CoreSight™ Architecture Specification* (ARM IHI 0029).
- *ARM® CoreSight™ SoC-400 Technical Reference Manual* (ARM DDI 0480).
- *ARM® AMBA® AXI and ACE Protocol Specification AXI3, AXI4, and AXI4-Lite, ACE and ACE-Lite* (ARM IHI 0022).

- *ARM® AMBA® 3 AHB-Lite Protocol Specification* (ARM IHI 0033).
- *ARM® AMBA® APB Protocol Specification* (ARM IHI 0024).
- *ARM® Generic Interrupt Controller Architecture Specification* (ARM IHI 0048).
- *ARM® CoreLink™ Level 2 Cache Controller L2C-310 Technical Reference Manual* (ARM DDI 0246).

The following confidential books are only available to licensees:

- *ARM® Cortex® -R7 MPCore Configuration and Sign-off Guide* (ARM DII 0251).
- *ARM® Cortex®-R7 MPCore Integration Manual* (ARM DIT 0030).
- *ARM® CoreSight™ SoC-400 Implementation Guide* (ARM DII 0267).
- *ARM® CoreSight™ SoC-400 Integration Manual* (ARM DIT 0037).
- *ARM® CoreSight™ SoC-400 System Design Guide* (ARM DGI 0018).
- *ARM® CoreSight™ SoC-400 User Guide* (ARM DUI 0563).

### Other publications

This section lists relevant documents published by third parties:

- *1149.1-2013 -IEEE Standard for Test Access Port and Boundary-Scan Architecture (JTAG).*
- *754-2008 - IEEE Standard for Floating-Point Arithmetic.*

## Feedback

ARM welcomes feedback on this product and its documentation.

### Feedback on this product

If you have any comments or suggestions about this product, contact your supplier and give:

- The product name.
- The product revision or version.
- An explanation with as much information as you can provide. Include symptoms and diagnostic procedures if appropriate.

### Feedback on content

If you have comments on content then send an e-mail to [errata@arm.com](mailto:errata@arm.com). Give:

- The title.
- The number, ARM DDI 0458C.
- The page numbers to which your comments apply.
- A concise explanation of your comments.

ARM also welcomes general suggestions for additions and improvements.

———— **Note** —————

ARM tests the PDF only in Adobe Acrobat and Acrobat Reader, and cannot guarantee the quality of the represented document when used with any other PDF reader.

---

# Chapter 1

## Introduction

This chapter introduces the Cortex-R7 MPCore processor and its features. It contains the following sections:

- *About the Cortex-R7 MPCore processor* on page 1-2.
- *Compliance* on page 1-3.
- *Features* on page 1-4.
- *Interfaces* on page 1-5.
- *Configurable options* on page 1-6.
- *Redundant processor comparison* on page 1-8.
- *Test features* on page 1-9.
- *Product documentation and design flow* on page 1-10.
- *Product revisions* on page 1-12.

## 1.1 About the Cortex-R7 MPCore processor

The Cortex-R7 MPCore processor is a mid-range processor for use in deeply-embedded, real-time systems, and consists of one or two Cortex-R7 processors in a single MPCore device. It implements the ARMv7-R architecture, and includes Thumb-2 technology for optimum code density and processing throughput.

The pipeline has a dual *Arithmetic Logic Unit* (ALU), with dual-issuing of instructions for efficient utilization of other resources such as the register file. The processor has *Level 1* (L1) data cache coherency in a cluster with up to two processors, and an optional hardware *Accelerator Coherency Port* (ACP) is provided to reduce software cache maintenance operations when sharing memory regions with other masters.

Interrupt latency is kept low by interrupting and restarting load-store multiple instructions, and by use of an integrated interrupt controller. The Cortex-R7 MPCore processor provides two specialized memory solutions for low-latency and determinism:

- *Tightly-Coupled Memory* (TCM) offers very low latency and determinism, but has limited memory space.
- Local RAM, cached by L1, offers less low latency than TCM, but latencies are still bounded, so it is deterministic. This solution offers larger memory space than TCM, and is coherent in multiprocessor configurations.

Optional *Error Correcting Code* (ECC) is used on the processor ports and in L1 memories to provide improved reliability and address fault-critical applications.

Many of the features, including the caches, TCM, and ECC are configurable so that a given processor implementation can be tailored to the application for efficient power and area usage.

## 1.2 Compliance

The Cortex-R7 MPCore processor complies with, or implements, the specifications described in:

- [ARM architecture](#).
- [Trace macrocell](#).
- [Advanced Microcontroller Bus Architecture](#).
- [Debug architecture](#).
- [Generic Interrupt Controller architecture](#).

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

### 1.2.1 ARM architecture

The Cortex-R7 MPCore processor implements the ARMv7-R architecture and ARMv7 debug architecture. The ARMv7-R architecture provides 32-bit ARM and 16-bit and 32-bit Thumb instruction sets, including a range of *Single Instruction, Multiple Data (SIMD) Digital Signal Processing (DSP)* instructions that operate on 16-bit or 8-bit data values in 32-bit registers. The optional FPU implements the VFPv3-D16 architecture. This includes the VFPv3 instruction set. See *ARM® Architecture Reference Manual, ARMv7-A and ARMv7-R edition*.

### 1.2.2 Trace macrocell

The Cortex-R7 MPCore can include an optional *Embedded Trace Macrocell (ETM)*. This ETM implements ETM architecture version 4. See *ARM® CoreSight™ Embedded Trace Macrocell v4 Architecture Specification*.

### 1.2.3 Advanced Microcontroller Bus Architecture

The Cortex-R7 MPCore processor complies with the AMBA 3 protocol. See *ARM® AMBA® AXI Protocol Specification* and *ARM® AMBA® 3 APB Protocol Specification*.

### 1.2.4 Debug architecture

The Cortex-R7 MPCore processor implements version 7.1 of the ARM Debug architecture that includes support for Security Extensions and CoreSight. See the *ARM® CoreSight™ Architecture Specification*.

### 1.2.5 Generic Interrupt Controller architecture

The Cortex-R7 MPCore processor implements the ARM *Generic Interrupt Controller (GIC)* v1.0 architecture. See the *ARM® Generic Interrupt Controller Architecture Specification*.

## 1.3 Features

The Cortex-R7 MPCore processor includes the following features:

- One or two processors, with the following features:
  - Superscalar, variable-length, out-of-order pipeline.
  - Static and dynamic branch prediction. The dynamic branch prediction has *PREDictor* (PRED) RAM for the *Global History Buffer* (GHB), and an 8-entry return stack.
  - Support for both low interrupt latency mode and normal interrupt latency mode.
  - Non-maskable interrupt.
  - Optional *Floating Point Unit* (FPU) in each processor. If you choose to instantiate the FPU, there are two possible designs, either an optimized FPU, which is single-precision and half-precision, or a full FPU, which is single-precision, half-precision, and double-precision.
  - Each processor has a debug APB interface.
  - Optional *Memory Reconstruction Port* (MRP) in each processor for memory reconstruction.
  - A *Performance Monitoring Unit* (PMU) in each processor.
- A Harvard L1 memory system for each processor with:
  - An ARMv7-R architecture *Memory Protection Unit* (MPU) with 12 or 16 regions, each region with a minimum resolution of 256 bytes.
  - Optional data and instruction TCM, configurable from 0KB to 128KB.
  - Optional AXI slave ports to access TCMs from external hardware.
  - Optional instruction caches and data caches, with configurable sizes of 0KB, 4KB, 8KB, 16KB, 32KB, or 64KB.
  - Optional ECC support on the RAM arrays and external buses.
- A *Snoop Control Unit* (SCU) that connects one or two processors to the memory system through the AXI3 interfaces.
- An integrated *Generic Interrupt Controller* (GIC) supporting a configurable number of external IRQs, from 0 to 480 *Shared Peripheral Interrupts* (SPIs) in increments of 32. There are always 32 internal IRQ lines reserved for inter-processor interrupts, and timer interrupts.
- Integrated private timers, a watchdog timer, and a global timer.
- The ability to implement redundant processor logic, for example, in fault detection.
- High-speed *Advanced Microprocessor Bus Architecture* (AMBA) *Advanced eXtensible Interfaces* (AXI3):
  - A single 64-bit master AXI3 interface.
  - An optional 64-bit AXI slave *Accelerator Coherency Port* (ACP).
  - An optional second AXI3 master interface with address filtering support.
  - Memory-mapped 32-bit AXI3 peripheral port in each processor.
 Optional ECC protection on data and parity on control bits.
- Optional ETM interface with full instruction and data trace, with either:
  - One ETM, statically shared between the processors.
  - Two ETMs, one dedicated to each processor.



## 1.4 Interfaces

The Cortex-R7 MPCore processor has the following interfaces for external access:

- [APB Debug interface](#).
- [ETM/ATB interface](#).
- [PMU interface](#).
- [Test interface](#).

### 1.4.1 APB Debug interface

AMBA APBv3 is used for debugging purposes.

### 1.4.2 ETM/ATB interface

You can implement the Cortex-R7 MPCore processor with an optional ETM. If the ETM is present, there is a dedicated ETM and *AMBA Trace Bus* (ATB) interface, see [ETM/ATB interface signals on page A-37](#). The ETM provides instruction and data trace for the processor. For more information on the ETM itself, see the *ARM® CoreSight™ ETM-R7 Technical Reference Manual*.

The ETM/ATB interface conforms to the AMBA ATB specification. The ETM/ATB interface includes the following signals:

- Instruction trace interface.
- Data trace interface.

### 1.4.3 CTM interface

The processor implements a single cross trigger channel interface. This external interface is connected to the CoreSight *Cross Trigger Interface* (CTI) corresponding to each processor through a simplified *Cross Trigger Matrix* (CTM).

### 1.4.4 PMU interface

See [Performance Monitoring Unit on page 10-3](#) for more information about the event bus.

### 1.4.5 Test interface

The test interface provides support for test during manufacture of the Cortex-R7 MPCore processor using *Memory Built-In Self Test* (MBIST). For more information on the test interface, see [MBIST interface on page A-32](#). See the *ARM® Cortex®-R7 MPCore Integration Manual* for information about the timings of these signals.

## 1.5 Configurable options

Table 1-1 shows the features of the Cortex-R7 MPCore processor that can be configured using either build or pin configurations. See [Product documentation and design flow on page 1-10](#) for information about configuration of the processor. Many of these features, if included, can also be enabled and disabled during software configuration.

**Table 1-1 Configurable options**

Feature	Range of options	Sub-options	Build or pin configuration
Number of processors and optional redundancy	Single processor, no redundancy	One processor	Build
	Single processor, with redundancy	One processor built as lock-step	Build
	Dual processor, no redundancy	Two processors	Build
	Dual processor, redundant processor	Two processors built as split/lock, lock-step mode	Build and lock-step mode pin = 1
		Two processors built as split/lock, normal mode	Build and lock-step mode pin = 0
Instruction cache	No instruction cache <sup>a</sup>	-	Build
	Instruction cache included	No ECC <sup>b</sup> 64-bit ECC	Build
		4KB, 8KB, 16KB, 32KB, or 64KB	Build
Data cache	No data cache <sup>c</sup>	-	Build
	Data cache included	No ECC <sup>b</sup> 32-bit ECC	Build
		4KB, 8KB, 16KB, 32KB, or 64KB	Build
Instruction TCM	No Instruction TCM	-	Build
	Instruction TCM included	No ECC <sup>b</sup> 64-bit ECC	Build
		4KB, 8KB, 16KB, 32KB, 64KB, or 128KB	Build
Data TCM	No Data TCM	-	Build
	Data TCM included	No ECC <sup>b</sup> 32-bit ECC	Build
		4KB, 8KB, 16KB, 32KB, 64KB, or 128KB	Build
Branch Target Address Cache (BTAC) size	256, 512, 1024, 2048, or 4096 entries, default is 512	- <sup>b</sup>	Build
PREDictor (PRED) RAM size	1024, 2048, or 4096 entries, default is 4096	- <sup>b</sup>	Build

Table 1-1 Configurable options (continued)

Feature	Range of options	Sub-options	Build or pin configuration
FPU	Not included	-	Build
	FPU included	Single-precision implementation	Build
		Double-precision implementation	Build
MPU	Number of regions	12 region option	Build
		16 region option	Build
AXI master ports	1 or 2	1	Build
		2, with address filtering	Build and pin
ETM	Included or not	-	Build
<i>Memory Reconstruction Port (MRP)</i>	Included or not	-	Build
Support for ECC	Used or not	-	Build
Number of interrupts	0-480 in range of 32	-	Build

- a. If you select no instruction cache, you must also select no data cache.
- b. The ECC parameter is global for the instruction and data cache RAMs, and ITCM and DTCM.  
BTAC and PRED RAM are protected by parity, initiated using the same ECC parameter.
- c. If you select no data cache, you must also select no instruction cache.

## 1.6 Redundant processor comparison

You can implement the Cortex-R7 MPCore processor with a second, redundant copy of most of the logic. The redundant logic includes a second processor that shares the input pins and the cache and TCM of the master processor, so only one set of cache and TCM is required. The redundant logic includes the individual processor logic, but not the ETM logic if the ETM is present. The redundant logic operates in lock-step with the processor, but does not directly affect the processor behavior in any way. The master processor drives the output pins and the RAMs. The redundant logic also includes a copy of

- SCU logic. The SCU Tag RAM is not duplicated.
- AXI TCM slave, if TCM is present.

Comparison logic can be included at build time. This logic compares the outputs of the processor, SCU, and AXI TCM slave with those of their redundant copy. These comparators are enabled through the **COMPENABLE** input signal. If a fault occurs in either the main or redundant logic because of radiation or circuit failure, the comparison logic detects it and the output signal **COMPFAULT** is asserted. Used in conjunction with the RAM error detection schemes, this can help protect the system from faults. **COMPENABLE** can be asserted only after a initialization phase, see the *ARM® Cortex®-R7 MPCore Configuration and Sign-Off Guide*. See the *ARM® Cortex®-R7 MPCore Integration Manual* for more information about **COMPENABLE** and **COMPFAULT** or contact your system integrator.

ARM provides example comparison logic, but you can change this during implementation. If you are implementing a dual-redundant configuration, contact ARM for more information.

### 1.6.1 Split/lock

If two Cortex-R7 processors are included and the Split/lock infrastructure implemented, the processor group can operate in one of two modes:

<b>Split mode</b>	Operates as a multiprocessor configuration, with both processors capable of doing multiprocessing, by maintaining L1 data cache coherency. Each processor uses its dedicated cache RAM. Also known as performance mode.
<b>Locked mode</b>	Operates in lock-step mode. The second processor works as redundant logic for the individual processor logic, and the SCU logic, but not the ETM logic if the ETM is present. The processor 1 side cache RAM remains implemented but not used.

You can select the usage mode with the **SAFEMODE** input signal. This input can be changed only while the processor group is held in reset and must remain stable when out of reset.

For more information about how to make a change in processor mode, contact your system integrator.

If you are implementing a Split/lock configuration, contact ARM for more information.

## 1.7 Test features

The Cortex-R7 MPCore processor is delivered as fully-synthesizable RTL and is a fully-static design. Scan-chains and test wrappers for production test can be inserted into the design by the synthesis tools during implementation.

Production test of the processor cache and TCM RAMs can be done through the dedicated, pipelined MBIST interface. This interface shares some of the multiplexing present in the processor design, to improve the potential frequency compared to adding multiplexers to the RAM modules.

The TCM RAMs can be read and written directly by the program running on the processor. You can also use the dedicated AXI3 slave interface to access the TCMs. See [Accessing RAMs using the AXI3 interface on page 11-9](#) for more information about how to access the RAMs using the AXI3 slave interface.

## 1.8 Product documentation and design flow

This section describes the Cortex-R7 MPCore processor books and how they relate to the design flow.

See [Additional reading on page x](#) for more information about the books described in this section.

### 1.8.1 Documentation

The Cortex-R7 documentation is as follows:

#### Technical Reference Manual

The *Technical Reference Manual* (TRM) describes the functionality and the effects of functional options on the behavior of the Cortex-R7 MPCore processor. It is required at all stages of the design flow. The choices made in the design flow can mean that some behavior described in the TRM is not relevant. If you are programming the Cortex-R7 MPCore processor then contact:

- The implementer to determine:
  - The build configuration of the implementation.
  - What integration, if any, was performed before implementing the Cortex-R7 MPCore processor.
- The integrator to determine the pin configuration of the Cortex-R7 MPCore processor that you are using.

#### ————— Note —————

If the ETM is included, see the *ARM® CoreSight™ ETM-R7 Technical Reference Manual* for more information.

#### Configuration and Sign-off Guide

The *Configuration and Sign-off Guide* (CSG) describes:

- The available build configuration options and related issues in selecting them.
- How to configure the *Register Transfer Level* (RTL) description with the build configuration options.
- The processes to sign off the configured design.

The ARM product deliverables include reference scripts and information about using them to implement your design. Reference methodology flows supplied by ARM are example reference implementations. Contact your EDA vendor for EDA tool support.

The CSG is a confidential book that is only available to licensees.

#### Integration Manual

The *Integration Manual* (IM) describes how to integrate the Cortex-R7 MPCore processor into a SoC. It includes describing the pins that the integrator must tie off to configure the macrocell for the required integration. Some of the integration is affected by the configuration options used when implementing the Cortex-R7 MPCore processor.

The IM is a confidential book that is only available to licensees.

## 1.8.2 Design flow

The Cortex-R7 MPCore processor is delivered as synthesizable RTL. Before it can be used in a product, it must go through the following process:

### Implementation

The implementer configures and synthesizes the RTL to produce a hard macrocell. This might include integrating RAMs into the design.

**Integration** The integrator connects the implemented design into a SoC. This includes connecting it to a memory system and peripherals.

### Programming

This is the last process. The system programmer develops the software required to configure and initialize the processor, and tests the required application software.

Each process:

- Can be performed by a different party.
- Can include implementation and integration choices that affect the behavior and features of the Cortex-R7 MPCore 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 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 processor by programming particular values into registers. This affects the behavior of the processor.

---

#### Note

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 pin configuration options are selected. Reference to an enabled feature means one that has also been configured by software.

---

## 1.9 Product revisions

This section describes the differences in functionality between product revisions:

**r0p0** First release.

**r0p0-r0p1** The following changes have been made in this release:

- ID register values changed to reflect product revision status:  
**Main ID Register** 0x410FC171.
- Various engineering errata fixes.



# Chapter 2

## Functional Description

This chapter describes the functionality of the Cortex-R7 MPCore processor. It contains the following sections:

- *About the functions* on page 2-2.
- *Interfaces* on page 2-4.
- *Clocking, resets, and initialization* on page 2-5.
- *Power management* on page 2-13.
- *Processor ports* on page 2-19.

## 2.1 About the functions

The Cortex-R7 MPCore processor is a highly configurable processor for use in deeply-embedded systems. Figure 2-1 shows a Cortex-R7 MPCore processor design with two processors.

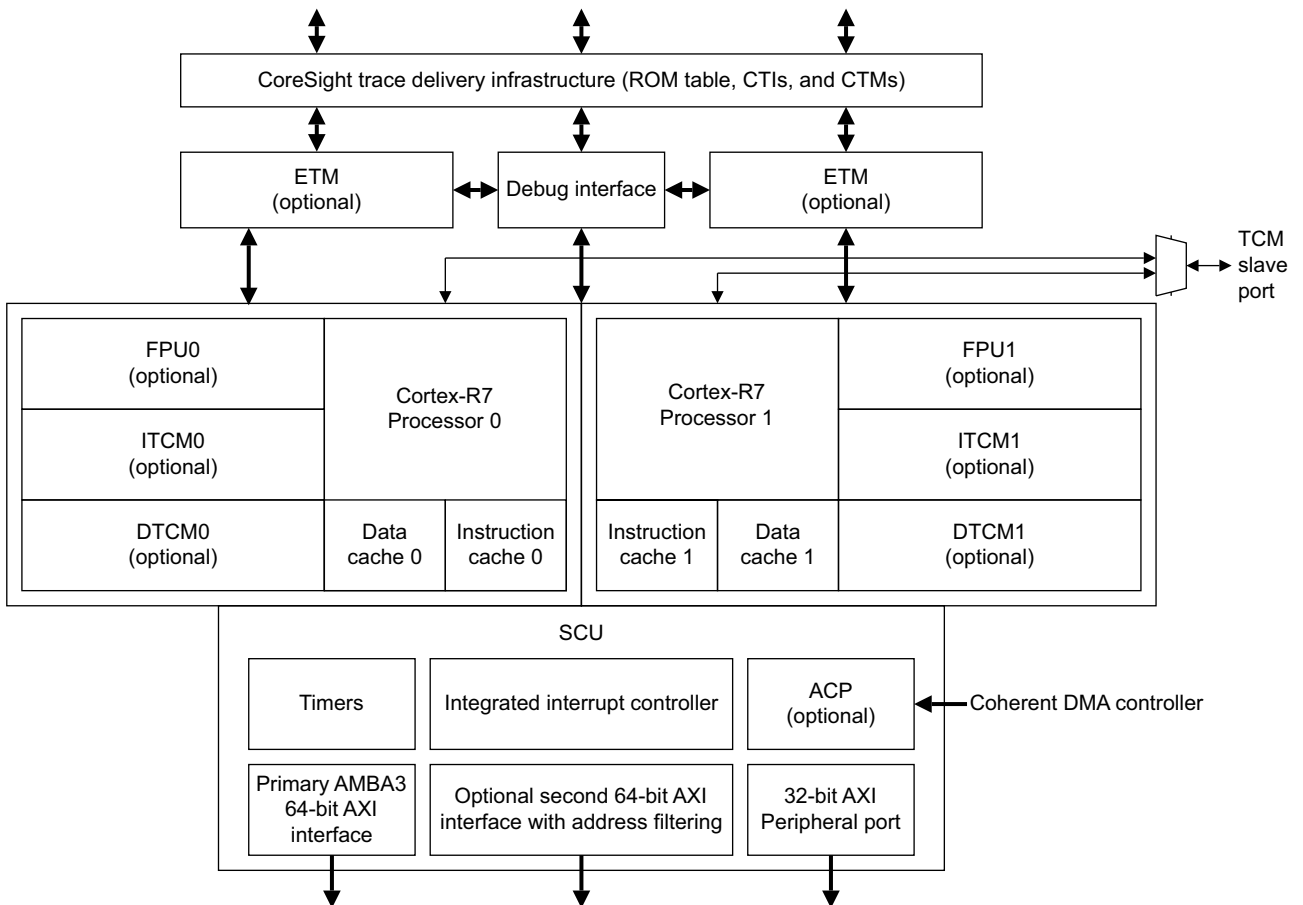


Figure 2-1 Cortex-R7 MPCore processor block diagram with two processors

### 2.1.1 Components of the Cortex-R7 MPCore processor

The main components of the Cortex-R7 MPCore processor are:

- [L1 memory system](#).
- [Snoop Control Unit on page 2-3](#).
- [Interrupt controller on page 2-3](#).
- [Timers on page 2-3](#).
- [Debug and Trace on page 2-3](#).
- [Split/lock on page 2-3](#).

#### L1 memory system

The L1 memory system has:

- 64-bit data paths throughout the memory system.
- Export of memory attributes for external memory systems.
- Separate optional instruction and data caches each with a fixed line length of 32 bytes.

## Snoop Control Unit

The SCU connects the Cortex-R7 processors to the memory system and peripherals through the AXI3 interfaces. The SCU has an optional AXI3 64-bit slave port, the *Accelerator Coherency Port* (ACP). See [Accelerator Coherency Port on page 2-23](#).

See [About multiprocessing and the SCU on page 9-2](#) for more information.

---

### Note

---

The SCU supports L1 data cache coherency, but does not support hardware management of coherency of the instruction caches.

---

## Interrupt controller

The interrupt controller provides support for handling multiple interrupt sources. See [Interrupt controller on page 9-19](#) for more information.

## Timers

The Timers provide the ability to schedule events and trigger interrupts. See [Private timer and watchdog on page 9-29](#) and [Global timer on page 9-35](#) for more information.

## Debug and Trace

The debug and trace logic includes:

- Support for ARMv7 Debug architecture with an APB slave interface for access to the debug registers.
- *Performance Monitor Unit* (PMU) based on the PMUv1 architecture.
- *Embedded Trace Macrocell* (ETM) based on the ETMv4 architecture and dedicated ATB interfaces per processor.
- *Cross Trigger Interface* (CTI) and *Cross Trigger Matrix* (CTM) for multi-processor debugging.

See the following for more information:

- [Performance Monitoring Unit on page 10-3](#).
- [Embedded Trace Macrocell on page 10-11](#).
- [Debug on page 10-12](#).

## Split/lock

The Cortex-R7 MPCore processor can be configured so that it can be switched, under reset, between a twin-processor performance mode and a dual-redundant lock-step. This feature imposes extra constraints on the software usage model. ARM provides an init code sequence to guarantee the processor and the redundant processor leave reset in exactly the same state. See [Static split/lock on page 7-16](#) for more information.

## 2.2 Interfaces

The Cortex-R7 MPCore processor has the following external interfaces:

- [AXI3 interface](#).
- [Debug and PMU APB interface](#).
- [ATB interface](#).
- [DFT interface](#).
- [MBIST controller interface](#).

### 2.2.1 AXI3 interface

The Cortex-R7 MPCore processor implements the AMBA 3.0 AXI with:

- A primary master port interface.
- An optional secondary master port interface, depending on the build configuration.
- A peripheral master port interface.
- A ACP slave port, depending on the build configuration.
- A TCM slave port, depending on the build configuration.

See [Processor ports on page 2-19](#).

### 2.2.2 Debug and PMU APB interface

The Cortex-R7 MPCore processor implements an APB slave interface that enables access to the following:

- Debug and PMU registers.
- CoreSight components (local ROM table, CTIs, and CTMs).
- ETM.

See the *ARM® CoreSight™ Architecture Specification* for more information.

### 2.2.3 ATB interface

The Cortex-R7 MPCore processor implements two dedicated ATB interfaces for each processor that output trace information for debugging. The ATB interfaces are compatible with the CoreSight architecture. See the *ARM® CoreSight® ETM Architecture Specification v4* for more information. See also [ETM/ATB interface signals on page A-37](#).

### 2.2.4 DFT interface

The Cortex-R7 MPCore processor implements a *Design For Test* (DFT) interface that enables an industry standard *Automatic Test Pattern Generation* (ATPG) tool to test logic outside of the embedded memories.

### 2.2.5 MBIST controller interface

The MBIST controller interface provides support for manufacturing testing of the memories embedded in the Cortex-R7 MPCore processor. MBIST is the industry standard method of testing embedded memories. MBIST works by performing sequences of reads and writes to the memory based on test algorithms. See [MBIST interface on page A-32](#) for information on this interface.

## 2.3 Clocking, resets, and initialization

The following sections describe clocking, resets, and initialization:

- [Clocking](#).
- [Resets](#).
- [Initialization on page 2-9](#).

### 2.3.1 Clocking

This section describes the following Cortex-R7 MPCore processor clocks:

- [CLK](#).
- [PERIPHCLK](#).
- [PERIPHCLKEN](#).

#### CLK

This is the main clock of the Cortex-R7 MPCore processor. All the processors in the device and the SCU are clocked with a distributed version of **CLK**. It is also the main clock for the ETMs and the local CoreSight infrastructure.

#### PERIPHCLK

The interrupt controller, global timer, private timers, and watchdog are clocked with **PERIPHCLK**. **PERIPHCLK** must be synchronous with **CLK**, and the **PERIPHCLK** clock period,  $N$ , must be configured as a multiple of the **CLK** clock period. This multiple  $N$  must be equal to, or greater than two.

#### PERIPHCLKEN

This is the clock enable signal for the interrupt controller and timers. The **PERIPHCLKEN** signal is generated at **CLK** clock speed. **PERIPHCLKEN** HIGH on a **CLK** rising edge indicates that there is a corresponding **PERIPHCLK** rising edge.

[Figure 2-2](#) shows an example of clocking the peripherals using these enable signals at a 3:1 ratio.

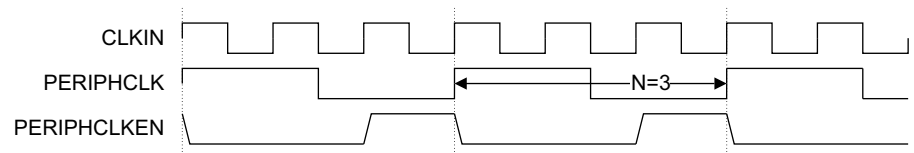


Figure 2-2 Clocking example on MPCore peripherals

#### DUALPERIPHCLK and DUALPERIPHCLKEN

This clock and clock enable signal are present if lock-step or split/lock is implemented. See [Clock and control signals on page A-3](#) for more information on how these signals are used.

### 2.3.2 Resets

The Cortex-R7 MPCore processor has the following reset signals, where  $N$  is 1 or 2:

- **CPUCLKOFF[N-1:0]**.
- **nCPURESET[N-1:0]**.
- **nDBGRESET[N-1:0]**.

- **DBGCLKOFF[N-1:0].**
- **nPERIPHRESET.**
- **nSCURESET.**
- **nCPUHALT[N-1:0].**
- **nWDRESET[N-1:0].**
- **WDRESETREQ[N-1:0].**

The following reset signals are present for CoreSight debug logic:

- **nCTRESET.**
- **CTCLKOFF.**

The following reset signals are present if ETM0 is present:

- **nETM0RESET.**
- **ETM0CLKOFF.**

The following reset signals are present if ETM1 is present:

- **nETM1RESET.**
- **ETM1CLKOFF.**

The following additional reset signals are present if lock-step or split/lock is implemented:

- **SCUCLKOFF.**
- **PERIPHCLKOFF.**
- **DUALPERIPHCLKOFF.**

The reset signals in the Cortex-R7 MPCore processor design enable you to reset different parts of the design independently. [Table 2-1](#) shows the supported reset combinations in a Cortex-R7 system. [n] refers to the processor that receives a reset.

**Table 2-1 Reset combinations in a Cortex-R7 system**

Reset signals	Cortex-R7 MPCore		Individual processors		Cortex-R7 MPCore debug	Individual processors	
	Powerup	Software	Powerup	Software		Debug	Watchdog flag
<b>nSCURESET and nPERIPHRESET</b>	0	0	1	1	1	1	1
<b>nCPURESET[1:0]</b>	All 0	All 0	[n]=0	[n]=0	All 1	All 1	All 1
<b>nDBGRESET[1:0]</b>	All 0	All 1	[n]=0	All 1	All 0	[n]=0	All 1
<b>nWDRESET[1:0]</b>	All 0	All 0	[n]=0 or all 1	[n]=0 or all 1	All 1	All 1	[n]=0

See [Clock and control signals on page A-3](#) and [Reset signals on page A-5](#).

The following sections describe the reset sequences:

- [Cortex-R7 MPCore powerup reset on page 2-7.](#)
- [Individual processor powerup reset on page 2-8.](#)
- [Individual processor software reset on page 2-8.](#)
- [Cortex-R7 MPCore debug reset on page 2-9.](#)
- [Individual processor debug reset on page 2-9.](#)
- [Individual processor watchdog flag reset on page 2-9.](#)

## Cortex-R7 MPCore powerup reset

You must apply powerup or cold reset to the Cortex-R7 MPCore processor when power is first applied to the system. In the case of powerup reset, the leading edge, that is the falling edge, of the reset signals do not have to be synchronous to **CLK**, but the rising edge must be. You must assert the reset signals for at least 10 **CLK** cycles to ensure correct reset behavior.

You must generate **DUALPERIPHCLK**, **DUALPERIPHCLKEN**, and **DUALPERIPHCLKOFF** as delayed versions of the equivalent primary core signals. Use the same delay as between the primary core and the redundant core.

ARM recommends the following reset sequence:

1. Apply **nCPURESET**, **nDBGRESET**, **nWDRESET**, **nSCURESET**, **nPERIPHRESET**, **nCTRESET**, **nETM0RESET** if ETM0 is present, and **nETM1RESET** if ETM1 is present.
2. Wait for at least 10 **CLK** cycles, or more if required by other components. There is no harm in applying more clock cycles than this, and maximum redundancy can be achieved by, for example, applying 15 cycles on every clock domain.
3. Stop the **CLK** clock input to the Cortex-R7 MPCore processor. You can use an integrated clock gating cell driven by a reset controller to stop the **CLK**.
4. Wait for the equivalent of approximately 10 cycles, depending on your implementation. This compensates for clock and reset tree latencies.
5. Release all resets.
6. Wait for the equivalent of approximately 10 cycles to compensate for clock and reset tree latencies.
7. Restart the **CLK** clock input.

For a lock-step or split/lock implementation, use the following reset sequence:

1. Apply **nCPURESET**, **nDBGRESET**, **nWDRESET**, **nSCURESET**, **nPERIPHRESET**, **nCTRESET**, **nETM0RESET** if ETM0 is present, and **nETM1RESET** if ETM1 is present.
2. Wait for at least 10 **CLK** cycles, or more if required by other components. There is no harm in applying more clock cycles than this, and maximum redundancy can be achieved by, for example, applying 10 cycles on every clock domain.
3. Drive **DUALPERIPHCLKOFF**, **SCUCLKOFF**, and **PERIPHCLKOFF** HIGH.
4. Stop the **CLK** clock input to the Cortex-R7 MPCore processor.
5. Wait for the equivalent of approximately 10 cycles, depending on your implementation. This compensates for clock and reset tree latencies.
6. Release all resets.
7. Wait for the equivalent of approximately 10 cycles, to compensate for clock and reset tree latencies.
8. Restart the **CLK** clock and maintain **DUALPERIPHCLKOFF** HIGH.  
**SCUCLKOFF** and **PERIPHCLKOFF** are driven LOW. If you want to, you can maintain these two signals HIGH a few more clock cycles but this is not required.
9. After  $P \times \text{CLK}$  cycles, after **PERIPHCLKOFF** is driven low, drive **DUALPERIPHCLKOFF** LOW.

---

**Note**

---

P\* is defined by the number of delay cycles introduced between the main processor and the dual-redundant processor. It is configurable and implementation defined. The default value used in the Cortex-R7 MPCore processor is 2.

---

**Individual processor powerup reset**

This reset is for the processor 0 or processor 1 level. It initializes the whole logic in a single processor, including its debug logic. It is expected to be applied when this individual processor exits from powerdown or dormant state. This reset only applies to configurations where each individual processor is implemented in its own power domain.

The reset sequence is as follows:

1. Apply **nCPURESET[n]** and **nDBGRESET[n]**. You can apply the **nWDRESET[n]** reset if you want to reset the corresponding watchdog flag.
2. Wait for at least 10 **CLK** cycles, or more if required by other components. There is no harm in applying more clock cycles than this.
3. Assert **CPUCLKOFF[n]** and **DBGCLKOFF[n]** with a value of 1'b1.
4. Wait for the equivalent of approximately 10 cycles, depending on your implementation. This compensates for clock and reset tree latencies.
5. Release all resets.
6. Wait for the equivalent of approximately 10 cycles, to compensate for clock and reset tree latencies.
7. Deassert **CPUCLKOFF[n]** and **DBGCLKOFF[n]**. This ensures that all registers in the processor see the same **CLK** edge on exit from the reset sequence.

The individual processor powerup reset can be extended to enable its corresponding ETM to be powered down. To wake up from powerdown or dormant mode with ETM, use the **nETM[n]RESET** and **ETM[n]CLKOFF** signals in the same way as **nCPURESET[n]** and **CPUCLKOFF[n]**.

---

**Note**

---

When resetting a processor from a powerdown state while the ETM is not reset, ARM recommends that you disable the ETM using the APB port before resetting the processor. This avoids the risk of tracing bad data at the point when the processor is reset.

---

**Individual processor software reset**

This reset is for the processor 0 or processor 1 level, without debug logic. It initializes all functional logic in a single individual processor apart from its debug logic. All breakpoints and watchpoints are retained during this individual warm reset. This reset only applies to configuration where each individual processor is implemented in its own power domain.

ARM recommends that you use the reset sequence described in *Individual processor powerup reset*, except that **nDBGRESET**, **nCTIRESET**, and **nETMxRESET** must not be asserted during the sequence. This ensures that the debug registers of the individual processors retain their values.



### Cortex-R7 MPCore debug reset

This reset initializes the debug logic in all processors present in the cluster. To perform a Cortex-R7 MPCore debug reset, assert all **nDBGRESET** signals for a number of **CLK** cycles. **CPUCLKOFF**, and **ETM[0/1]CLKOFF** if the ETM is present, must remain deasserted during this reset sequence.

### Individual processor debug reset

This reset initializes the debug logic in an individual processor in the cluster. To perform an individual processor debug reset, assert the corresponding **nDBGRESET[n]** signal for a number of **CLK** cycles. **CPUCLKOFF**, and **ETM[n]CLKOFF** if the ETM is present, must remain deasserted during this reset sequence.

---

#### Note

When lock-step or split/lock is implemented, the software must clear the INTdis bit of the *Debug Status and Control Register* (DBGDSCR) before applying a debug reset. This is because if a pending interrupt is masked by the INTdis bit and a debug reset occurs, this bit is cleared by the reset and the interrupt is taken immediately by both processors on the same clock cycle.

---

### Individual processor watchdog flag reset

This reset clears the watchdog flag associated with a single individual processor. Watchdog functionality is independent of all other processor functionality, so this reset is independent of the all other resets.

---

#### Note

When a watchdog reset request occurs and when lock-step or split/lock is implemented, the reset must not be applied immediately to the entire Cortex-R7 MPCore processor. This is because after reset the sticky flag is set in one processor, but not the other and this leads to the assertion of **COMPFAULT**. Therefore, if one processor has its watchdog flag set, the other processor must reach the same state, that is, also having its watchdog flag set. This can be controlled using the **PERIPHCLKOFF** and **DUALPERIPHCLKOFF** signals.

---

## 2.3.3 Initialization

Most of the architectural registers in the Cortex-R7 MPCore processor, such as r0-r14, and s0-s31 and d0-d15 when floating-point is included, have an unknown value after reset. Because of this, you must initialize these for all modes before they are used, using an immediate-MOV instruction, or a PC-relative load instruction. The *Current Program Status Register* (CPSR) is given a known value on reset. This is described in the *ARM® Architecture Reference Manual ARMv7-A and ARMv7-R edition*. The reset values for the CP15 registers are described along with the registers in [Chapter 4 System Control](#).

In addition, before you run an application, you might want to:

- Program particular values into various registers, for example, stack pointers.
- Enable various features, for example, error correction.
- Program particular values into memory, for example, the TCMs.

The following sections describe other initialization requirements:

- [MPU on page 2-10](#).
- [FPU on page 2-10](#).
- [Caches on page 2-10](#).

- [TCM](#).

## MPU

Before you can use MPU you must:

- Program and enable at least one of the regions.
- Enable the MPU in the SCTL. See [System Control Register on page 4-23](#).

Do not enable the MPU unless at least one MPU region is programmed and active. If the MPU is enabled, before using the TCM interfaces you must program MPU regions to cover the TCM regions to give access permissions to them.

## FPU

If the Cortex-R7 MPCore processor has been built with an FPU, you must enable it before *Vector Floating Point* (VFP) instructions can be executed. Enable the FPU as follows:

- Enable access to the FPU in the coprocessor access control register. See [Coprocesor Access Control Register on page 4-27](#)
- Enable the FPU by setting the EN-bit in the FPEXC register, see [Floating-Point Exception Register on page 5-10](#).

## Caches

If the Cortex-R7 MPCore processor has been built with instruction or data caches, they must be invalidated before they are enabled, otherwise UNPREDICTABLE behavior can occur. See [About the caches on page 6-8](#).

An invalidate all operation never reports any ECC errors. See [Auxiliary Control Register on page 4-25](#).

## TCM

The Cortex-R7 MPCore processor does not initialize the TCM RAMs, so you must initialize all the TCMs. In addition, you might want to preload instructions or data into the TCM for the main application to use. This section describes various ways that you can perform data preloading. You can also configure the Cortex-R7 MPCore processor to use the TCMs from reset. See [Instruction and data TCM on page 8-15](#), [DTCM Region Register on page 4-33](#) and [ITCM Region Register on page 4-34](#).

### Preloading TCMs

You can write data to the TCMs using either store instructions or the AXI3 slave interface. Depending on the method you choose, you might require:

- Particular hardware on the SoC that you are using.
- Boot code.
- A debugger connected to the processor.

Methods to preload TCMs include:

### Memory copy with running boot code

The 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 do this, and it might be necessary to give the TCM one base address while the copy is occurring, and a different base address when the application is being run.

### Copy data from the debug communications channel

The boot code includes a routine to read data from the *Debug Communications Channel* (DCC) and write it into the TCM. The debug host feeds the data for this operation into the DCC by writing to the appropriate registers on the processor APB debug port.

### Execute code in debug halt state

The debug host puts the Cortex-R7 MPCore processor into debug halt state and then feeds instructions into it through the *Instruction Transfer Register* (DBGITR). The Cortex-R7 MPCore processor executes these instructions, that replace the boot code in either of the two methods previously described.

### DMA into TCM

The SoC includes a *Direct Memory Access* (DMA) device that reads data from a ROM, and writes it to the TCMs through their AXI slave interfaces.

### Preloading TCMs with ECC

The error code bits in the TCM RAM, if configured with an error scheme, are not initialized by the Cortex-R7 MPCore processor. Before a RAM location is read with ECC enabled, the error code bits must be initialized. To calculate the error code bits correctly, the logic must have all the data in the data chunk that those bits protect. Therefore, when the TCM is being initialized, the writes must be of the same width and aligned to the data chunk that the error scheme protects.

You can initialize the TCM RAM with error checking turned on or off, according to the following rules. See [Auxiliary Control Register on page 4-25](#). You can use the **ITCMECCEN** signal to enable the ITCM when leaving reset.

If the slave port is used, for TCM memory accesses, the transactions must start at a 32-bit aligned address for data or 64-bit aligned address for instructions, and read or write a continuous block of memory, containing a multiple of 4 bytes for data or 8 bytes for instruction. All bytes in the block must be written, that is, have their byte lane strobe asserted.

If initialization is done by running code on the Cortex-R7 MPCore processor, this is best done by a loop of stores that write to the whole of the TCM memory as follows:

- If the scheme is 32-bit ECC, use *Store Word* (STR), *Store Two Words* (STRD), or *Store Multiple Words* (STM) instructions to 32-bit aligned addresses.
- If the scheme is 64-bit ECC, use STRD or STM, that has an even number of registers in the register list with a 64-bit aligned starting address.

#### ———— Note ————

You can use the alignment-checking features of the Cortex-R7 MPCore processor to help you ensure that memory accesses are 32-bit aligned, but there is no checking for 64-bit alignment. If you are using STRD or STM, an alignment fault is generated if the address is not 32-bit aligned. For the same behavior with STR instructions, enable strict-alignment-checking by setting the A-bit in the System Control Register. See [System Control Register on page 4-23](#).

### Using TCMs from reset

You can pin-configure the Cortex-R7 MPCore processor to enable the TCM interfaces from reset and to select the address at which each TCM appears from reset. This enables you to configure the processor to boot from TCM but, to do this, the TCM must first be preloaded with the boot code. The **nCPUHALT** input can be asserted while the processor is in reset to prevent the processor from fetching and executing instructions after coming out of reset. While the

processor is halted in this way, the TCMs can be preloaded with the appropriate data. When the **nCPUHALT** input is deasserted, the processor starts fetching instructions from the reset vector address in the normal way.

---

**Note**

---

When **nCPUHALT** has been deasserted to start the processor fetching, it must not be asserted again except when the processor is under processor or powerup reset. See [Resets on page 2-5](#).

---

## 2.4 Power management

The Cortex-R7 MPCore processor provides mechanisms and support to control both dynamic and static power dissipation. The following sections describe:

- [Individual processor power management](#).
- [Power domains on page 2-18](#).

### 2.4.1 Individual processor power management

Placeholders for clamps are inserted around each processor to simplify implementation of different power domains. See the *ARM® Cortex®-R7 MPCore Configuration and Sign-off Guide* for implementation information about the different power domains and the signals that require specific clamp values. Software is responsible for signaling to the Snoop Control Unit and the interrupt controller that a processor is shut off so that the processor can be seen as non-existent in the cluster. [Table 2-2](#) shows the power modes and the wake-up mechanisms for each mode:

**Table 2-2 Processor power modes**

Mode	Description	Wake-up mechanism
Run	Everything is clocked and powered-up.	-
Standby	The processor clock is stopped. Only logic required for wake-up is still active.	Standard standby mode wake up events.
Standby mode with RAM retention	The processor clock is stopped. Only logic required for wake-up is still active. RAMs are in retention.	
Dynamic RAM retention	Some RAM arrays are put in retention dynamically, that is, if they are not used or are temporarily disabled.	Normal request from processor logic.
Dormant	Everything is powered off except RAM arrays that are in retention mode.	External wake-up event to the power controller, that can perform a reset of the processor.
Shutdown	Everything is powered-off.	

Entry to Dormant or powered-off mode must be controlled through an external power controller. The CPU Power Status Register in the SCU is used in conjunction with CPU WFI entry flag to signal to the power controller the power domain that it can cut, using the PWRCTL bus. See [SCU CPU Power Status Register on page 9-9](#).

Entry to or exit from Standby mode with RAM retention or dynamic RAM retention must be controlled through an external power controller. See the *ARM® Cortex®-R7 MPCore Configuration and Sign-off Guide* for more information on the interface for RAM retention.

The following sections describe the processor power modes and the power management controller:

- [Run mode on page 2-14](#).
- [Standby modes on page 2-14](#).
- [Standby mode with RAM retention on page 2-15](#).
- [Dynamic RAM retention on page 2-16](#).
- [Dormant mode on page 2-16](#).
- [Shutdown mode on page 2-17](#).
- [Communication to the power management controller on page 2-17](#).

## Run mode

Run mode is the normal mode of operation, where all of the functionality of the processor is available. Everything is clocked and powered-up.

## Standby modes

There are two standby modes in Cortex-R7 processors:

- *Wait for Interrupt.*
- *Wait for Event.*

### Wait for Interrupt

*Wait for Interrupt* (WFI) is a feature of the ARMv7-R architecture that puts the processor in idle mode by disabling most of the clocks in the processor while keeping the processor powered up. Only the logic required for wake-up is still active. This reduces the power drawn to the static leakage current, leaving a small clock power overhead to enable the processor to wake up from WFI mode. A processor enters WFI mode by executing the WFI instruction.

When executing the WFI instruction, the processor waits for all instructions in the processor to complete before entering the idle mode.

While the processor is in WFI mode, the clocks in the processor are temporarily enabled without causing the processor to exit WFI mode, when any of the following events are detected:

- A snoop request that must be serviced by the processor L1 data cache.
- An APB access to the debug or trace registers residing in the processor power domain.
- An AXI TCM slave port access can also temporarily enable the processor without causing the processor to exit WFI mode.

Exit from WFI mode occurs when the processor detects a reset or one of the WFI wake up events as described in the *ARM Architecture Reference Manual*. CP15 broadcasting operations also force an exit from WFI mode.

On entry into WFI mode, **STANDBYWFI** for that processor is asserted. Assertion of **STANDBYWFI** guarantees that the processor is in idle mode.

#### ———— Note ————

If the ETM is present, the **ETMACTIVE0/1** primary output must be taken into account to ensure the ETM has completed tracing.

**STANDBYWFI** continues to assert even if the clocks in the processor are temporarily enabled because of:

- A L1 Cache Controller snoop request, if present.
- An APB access.
- An AXI TCM slave port request.

### Wait for Event

*Wait for Event* (WFE) is a feature of the ARMv7-R architecture that uses a locking mechanism based on events to put the processor in idle mode by disabling most of the clocks in the processor while keeping the processor powered up. This reduces the power drawn to the static leakage current, leaving a small clock power overhead to enable the processor to wake up from WFE mode.

A processor enters WFE mode by executing the WFE instruction. When executing the WFE instruction, the processor waits for all instructions in the processor to complete before entering the idle mode. The WFE instruction ensures that all explicit memory accesses, that are prior to the WFE instruction in program order, have completed.

While the processor is in WFE mode, the clocks in the processor are temporarily enabled without causing the processor to exit out of WFE mode, when any of the following events are detected:

- A snoop request that must be serviced by the processor L1 data cache.
- An APB access to the debug or trace registers residing in the processor power domain.
- An AXI TCM slave port access can also temporarily enable the processor without causing the processor to exit WFI mode.

Exit from WFE mode occurs when the processor detects a reset, an AXI TCM slave port request, the assertion of the **EVENTI** input signal, or one of the WFE wake up events as described in the *ARM® Architecture Reference Manual ARMv7-A and ARMv7-R edition*. CP15 broadcasting operations also force an exit from WFE mode.

On entry into WFE mode, **STANDBYWFE** for that processor is asserted. Assertion of **STANDBYWFE** guarantees that the processor is in idle mode. **STANDBYWFE** continues to assert even if the clocks in the processor are temporarily enabled because of:

- An L1 Cache Controller snoop request, if present.
- An APB access.
- An AXI TCM slave port request.

### Standby mode with RAM retention

The processor logic is in WFI mode, and the RAM arrays are in retention mode. The RAM can be:

- The entire instruction cache.
- The entire data cache.
- The entire ITCM.
- The entire DTCM.
- The Prediction RAMs, BTAC and PRED.
- The SCU Tag RAM, if both processors are in WFI.

A WFI instruction must be executed. The **STANDBYWFI** primary output indicates the WFI mode. A temporary wake-up of the RAM can happen:

- On L1 data cache when a snoop coherency request occurs.
- On DTCM when an AXI TCM slave port request to the data TCM occurs.
- On ITCM when an AXI TCM slave port request to the instruction TCM occurs.

To avoid waking up the data cache when the processor is in WFI, you can exclude the individual processor from the coherency domain. To do this, the state of the processor must be saved in the same way as when entering Dormant mode. The processor then indicates to the power controller that the device is ready to be powered down in the same way as when entering Dormant mode. The external power controller can then put the RAM in retention.

For information about the entry and exit signals, and the protocol to handle RAM retention, see the *ARM® Cortex®-R7 MPCore Configuration and Sign-off Guide*.

## Dynamic RAM retention

Some RAM arrays are put in retention dynamically, that is, if they are not used or are temporarily disabled. The RAM can be:

- The entire instruction cache or data cache.
- The entire ITCM or DTCM.
- A subset of the TCMs, using a number of address bits to select the address range of the TCMs.

---

### Note

---

The Prediction RAMs, that is, BTAC and PRED, and the SCU Tag RAM are not affected in this mode.

---

The external power controller determines when to put the RAMs in retention, and when to wake them up.

For information about the entry and exit signals, and the protocol to handle RAM retention, see the *ARM® Cortex®-R7 MPCore Configuration and Sign-off Guide*.

## Dormant mode

Dormant mode is designed to enable an individual processor to be powered down, while leaving the RAMs powered up and maintaining their state.

The RAM blocks that are to remain powered up must be implemented on a separate power domain, and there is a requirement to clamp all of the inputs to the RAMs to a known logic level, with the chip enable being held inactive. This clamping is not implemented in gates as part of the default synthesis flow because it would contribute to a tight critical path. Implementations that want to implement Dormant mode must add these clamps around the RAMs, either as explicit gates in the RAM power domain, or as pull-down transistors that clamp the values while the processor is powered down. All RAM blocks must remain powered up during Dormant mode.

Before entering Dormant mode, the state of the processor, excluding the contents of the RAMs that remain powered up in dormant mode, must be saved to external memory. These state saving operations must ensure that the following occur:

- All ARM registers, including CPSR and SPSR registers are saved.
- All system registers are saved.
- All debug-related state must be saved.
- The processor must correctly set the CPU Power Status Register in the SCU so that it enters Dormant Mode. See [SCU CPU Power Status Register on page 9-9](#).
- A Data Synchronization Barrier instruction is executed to ensure that all state saving has been completed.
- The processor then communicates with the power controller that it is ready to enter dormant mode by performing a WFI instruction so that power control output reflects the value of SCU CPU Power Status Register. See [SCU CPU Power Status Register on page 9-9](#).

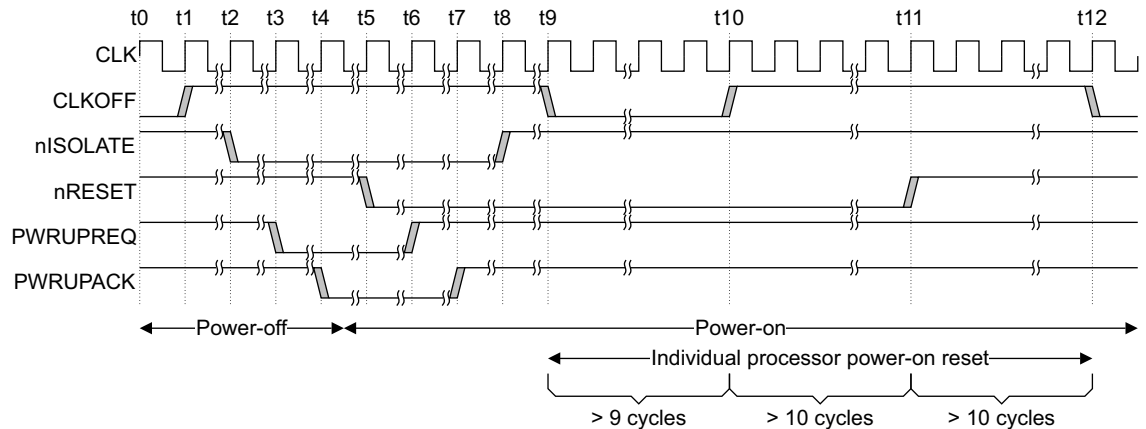


**Note**

If the ETM is present, the **ETMACTIVE0/1** primary output must be taken into account to ensure that the ETM has completed tracing before powering-off the processor or the ETM.

Transition from Dormant mode to Run mode is triggered by the external power controller. The external power controller must assert reset to the processor until the power is restored. After power is restored, the processor leaves reset and, by interrogating the power control register in the SCU, can determine that the saved state must be restored.

Figure 2-3 shows the powerdown and powerup sequence.



**Figure 2-3 Powerup and powerdown sequence**

### Shutdown mode

Shutdown mode has the entire device powered down, and all state, including cache, must be saved externally by software. The part is returned to the run state by the assertion of reset. This state saving is performed with interrupts disabled, and finishes with a DSB operation. The processor then indicates to the power controller that the device is ready to be powered down in the same way as when entering Dormant mode but, in this case, the processor must set the power mode in the SCU CPU Power Status Register to powerdown.

**Note**

If the ETM is present, the **ETMACTIVE0/1** primary output must be taken into account to ensure that the ETM has completed tracing before powering-off the processor or the ETM.

### Communication to the power management controller

Communication between the processor and the external power management controller can be performed using the **PWRCTLOn** Cortex-R7 MPCore output signals and Cortex-R7 MPCore input clamp signals.

The **PWRCTLOn** Cortex-R7 MPCore output signals constrain the external power management controller. The value of **PWRCTLOn** depends on the value of the SCU CPU Power Status Register. See [SCU CPU Power Status Register on page 9-9](#). The SCU CPU Power Status Register value is only copied to **PWRCTLOn** after the individual processor signals that it is ready to enter low power mode by executing a WFI instruction and subsequent **STANDBYWFI** output assertion.

## 2.4.2 Power domains

The Cortex-R7 MPCore processor can support the following power domains:

- One for each of the processors.
- One for each of the individual processor Cache RAM arrays, including the *Branch Predictor* (BP) RAMs.
- One for each of the individual processor TCM RAM arrays.
- One for the SCU duplicated tag RAMs.
- One for the remaining logic, the SCU logic cells, and private peripherals.

---

**Note**

If the ETM is included, each ETM for each processor has its own power domain. In addition, the local CoreSight logic, that is, CTI0 and CTI1, CTM, APB multiplexer, and ROM table, are also in a separate power domain.

---

Each power domain has its own clock and reset signal, and its own clock off signal. When a power domain is powered-off, some clamp values might be driven HIGH. See the *ARM® Cortex®-R7 MPCore Configuration and Sign-off Guide* for more information.

## 2.5 Processor ports

This section describes the following Cortex-R7 MPCore processor ports:

- [AXI master port 0](#).
- [AXI master port 1](#).
- [AXI peripheral port on page 2-20](#).
- [AXI TCM slave port on page 2-21](#).
- [Accelerator Coherency Port on page 2-23](#).
- [Memory Reconstruction Port on page 2-23](#).
- [Private memory region on page 2-23](#).

### 2.5.1 AXI master port 0

This port has optional ECC protection on data and parity on control bits, see [ECC on external AXI bus on page 7-7](#).

This port does not support AXI locked writes, that is, **AWLOCKM0[1]** is always 0.

This port supports five bits of AXI IDs, although AXI IDs can be larger if the ACP has more than four bits of ID. For example, if the ACP ID has 4 bits, there are 5 bits on the AXI master port. If the ACP ID has 8 bits, there are 9 bits on the AXI master port.

---

#### Note

- ID bit encoding is used to differentiate between different types of traffic happening in parallel. The encoding of the IDs is implementation-specific.
  - You can use the **AxUSER** buses to identify the origin of the traffic, that is, processor 0, processor 1, or the ACP.
- 

### 2.5.2 AXI master port 1

This port is optional, and has optional ECC protection on data and parity on control bits, see [ECC on external AXI bus on page 7-7](#). It has an address filtering feature enabled by the [SCU Control Register on page 9-6](#). When the master address filtering is enabled through the **MFILTEREN** input or the SCU Control Register, any access in the address range between the master filtering start address and the master filtering end address is issued on AXI master port 1. All other accesses outside of this range are directed onto AXI master port 0. The start and end addresses are configurable in the following SCU registers:

- [Master Filtering Start Address Register on page 9-11](#), where the value is defined by the **MFILTERSTART[11:0]** input when leaving reset.
- [Master Filtering End Address Register on page 9-11](#), where the value is defined by the **MFILTEREND[11:0]** input when leaving reset.

The granularity of the mapped memory is 1MB using the following formula:

Memory\_space (MB) = Start to End + 1.

This filtering rule is applied independently of the AXI request type and attributes.

When master address filtering is disabled, accesses can be issued on either AXI master port 0 or AXI master port 1, if the AXI ordering rules are respected. In this case, locked and exclusive accesses are always issued on AXI master port 0.

This port does not support locked writes, that is, **AWLOCKM1[1]** is always 0.

This port supports five bits of AXI IDs, although AXI IDs can be larger if the ACP has more than four bits of ID. For example, if the ACP ID has 4 bits, there are 5 bits on the AXI master port. If the ACP ID has 8 bits, there are 9 bits on the AXI master port.

---

**Note**

---

ID bit encoding is used to differentiate between different types of traffic happening in parallel. The encoding of the IDs is implementation-specific.

---

If you connect an L2 cache controller to AXI master port 0 and AXI master port 1, you cannot enable address filtering on AXI master port 1. There is no restriction on enabling address filtering on the AXI peripheral port. Some L2 cache controllers, such as the CoreLink Level 2 Cache Controller, can enable their own address filtering.

### 2.5.3 AXI peripheral port

The AXI peripheral port is a dedicated memory-mapped 32-bit AXI bus. It is used to access certain peripherals with a low latency, and having burst support. The memory mapping is done by address filtering.

This port does not support locked writes. **AWLOCKMP[1]** is always 0.

The AXI peripheral port does not support 64-bit accesses, and these accesses always abort. Processor cacheable accesses mapped to the peripheral port also abort, because they are doing linefill requests, that is, four 64-bit accesses. Normal memory accesses to the peripheral port also abort. These accesses abort because of their size, not because of their memory attributes.

Any access in the address range between the peripheral filtering start address and the peripheral filtering end address is issued on the AXI peripheral port. All other accesses outside of this range are directed onto the AXI master ports. The start and end addresses are configurable in the following SCU registers:

- [Peripherals Filtering Start Address Register on page 9-12](#), where the value is defined by **PFILTERSTART[11:0]** input when leaving reset.
- [Peripherals Filtering End Address Register on page 9-13](#), where the value is defined by **PFILTEREND[11:0]** input when leaving reset.

The granularity of the mapped memory is 1MB. This filtering rule is applied independently of the AXI request type and attributes. See [SCU registers on page 9-5](#) for more information.

This port supports five bits of AXI IDs, although AXI IDs can be larger if the ACP has more than four bits of ID. For example, if the ACP ID has 4 bits, there are 5 bits on the AXI peripheral port. If the ACP ID has 8 bits, there are 9 bits on the AXI peripheral port.

---

**Note**

---

- ID bit encoding is used to differentiate between different types of traffic happening in parallel. The encoding of the IDs is implementation-specific.
  - If the address filtering of the peripheral port and the address filtering of AXI master port 1 overlaps, the peripheral port has priority.
-

## 2.5.4 AXI TCM slave port

The AXI TCM slave port enables AXI masters, including the AXI master port of the processor if connected externally, to access data and instruction TCMs on the AXI system bus. You can use this for *Direct Memory Access* (DMA) to and from the TCM RAMs, and for software test of the TCM.

The AXI slave port accesses have lower priority than the *Load Store Unit* (LSU) or *PreFetch Unit* (PFU) accesses. The MPU does not check accesses from the AXI TCM slave.

The Cortex-R7 MPCore processor has a single AXI slave port. The port is 64 bits wide and conforms to the AXI standard. Within the AXI standard, the slave port uses the **AWUSERST** and **ARUSERST** signals as two separate chip select input signals to enable access as [Table 2-3](#) shows.

**Table 2-3 AXI TCM slave port access**

<b>AxUSERST[1:0]</b>	<b>Access</b>
00	Instruction TCM for processor 0
01	Data TCM for processor 0
10	Instruction TCM for processor 1
11	Data TCM for processor 1

The external AXI system must generate the chip select signals. The slave interface routes the access to the required RAM.

### Configurable AXI ID bits

You can configure the number of bits for the AXI IDs in the AXI TCM slave port at the implementation level. This number must be greater than or equal to 1 if the Cortex-R7 MPCore processor includes TCMs.

### Supported AXI transfers

[Table 2-4](#) to [Table 2-7](#) on [page 2-22](#) show the access that the AXI TCM slave port supports.

**Table 2-4 Doubleword accesses, aligned on 64-bit address**

<b>Signal</b>	<b>Access</b>
<b>AxSIZEST[2:0]</b>	0x3
<b>AxLENST[3:0]</b>	Any
<b>AxBURSTST[1:0]</b>	Any (FIXED, INCR, or WRAP)
<b>AxADDRST[2:0]</b>	0b000

**Table 2-5 Single word accesses, aligned on 32-bit address**

Signal	Access
<b>AxSIZEST[2:0]</b>	0x2
<b>AxLENST[3:0]</b>	0x0
<b>AxBURSTST[1:0]</b>	Ignored
<b>AxADDRST[2:0]</b>	00b0

**Table 2-6 Single halfword accesses, aligned on 16-bit address**

Signal	Access
<b>AxSIZEST[2:0]</b>	0x1
<b>AxLENST[3:0]</b>	0x0
<b>AxBURSTST[1:0]</b>	Ignored
<b>AxADDRST[2:0]</b>	00b0

**Table 2-7 Single byte accesses**

Signal	Access
<b>AxSIZEST[2:0]</b>	0x0
<b>AxLENST[3:0]</b>	0x0
<b>AxBURSTST[1:0]</b>	Ignored

Any other AXI accesses, such as unaligned or multiple accesses, result in a slave error, that is, **xRESPST[1:0]** = 0b10.

See [ECC on external AXI bus on page 7-7](#) for information about ECC errors.

The AXI TCM slave port also supports the following accesses, but with limited bandwidth:

- Doubleword accesses to the DTCM or ITCM where all byte strobes are not set, that is, **WSTRBST[7:0]** is not 0xFF.
- Word accesses to the DTCM where all byte strobes are not set, that is, **WSTRBST[7:0]** is not 0x0F or 0xF0.
- Word accesses to the ITCM, regardless of the byte strobe setting.
- Halfword accesses to the DTCM or ITCM, regardless of the byte strobe setting.
- Byte accesses to the DTCM or ITCM.

In the following cases, the AXI TCM slave port also gives a slave error:

- Non-single byte, halfword, or word accesses, that is, **AxSIZEST[2:0]** is not 0x3 and **AxLENST[3:0]** is not 0x0.
- The address exceeds the targeted TCM size, defined in **AxUSERST[1:0]**.
- The access is targeting a TCM of processor 1, that is, **AxUSERST[1]** = 1, when processor 1 is not present.

The **AxLOCKST**, **AxCACHEST**, and **AxPROTST** signals are not implemented on the AXI TCM slave port. This means that the AXI slave interface does not support locked or exclusive accesses. There is no exclusive monitor. If any master attempts an exclusive read, the AXI TCM slave port returns an OKAY response instead of an EXOKAY response. The master can treat this as an error condition indicating that the exclusive access is not supported. ARM recommends that the master does not perform the write portion of this exclusive operation.

### 2.5.5 Accelerator Coherency Port

This port is optional, and has optional ECC protection on data and parity on control bits, see [ECC on external AXI bus on page 7-7](#). The ACP is an AXI3 64-bit slave port that can be connected to non-cached AXI3 master peripherals, such as a DMA engine or cryptographic engine.

This AMBA 3 AXI compatible slave interface on the SCU provides an interconnect point for a range of system masters that, for overall system performance, power consumption, or to simplify software, are better interfaced directly with the Cortex-R7 MPCore processor.

See [Accelerator Coherency Port on page 9-39](#) and [Accelerator Coherency Port interface on page 11-12](#) for more information.

### 2.5.6 Memory Reconstruction Port

There is an MRP for each processor. All write accesses, regardless of their memory attributes, are exported from the Cortex-R7 MPCore processor through this port so that an image of the memory can be reconstructed. See [Memory Reconstruction Port on page 10-10](#) for more information.

### 2.5.7 Private memory region

All registers accessible by all processors within a Cortex-R7 MPCore design are grouped into two contiguous 4KB pages accessed through a dedicated internal bus. The base address of these pages is defined by the **PERIPHBASE[31:13]** inputs. See [Configuration signals on page A-7](#) for more information on **PERIPHBASE[31:13]**.

Global control registers and peripherals must be accessed through memory-mapped transfers to the private memory region.

Memory regions used for these registers must be marked as Device or Strongly-ordered in the MPU.

Access to the private memory region is little-endian only.

Access these registers with single load/store instructions. Load or store multiple accesses cause an abort to the requesting processor and the Fault Status Register shows this as a SLVERR.

[Table 2-8](#) shows the permitted access sizes for the private memory regions.

**Table 2-8 Permitted access sizes for private memory regions**

Private memory region	Permitted access sizes			
	Byte	Halfword <sup>a</sup>	Word <sup>b</sup>	Doubleword <sup>a</sup>
Global timer, private timers, and watchdogs	No	No	Yes	No

**Table 2-8 Permitted access sizes for private memory regions (continued)**

Private memory region	Permitted access sizes			
	Byte	Halfword <sup>a</sup>	Word <sup>b</sup>	Doubleword <sup>a</sup>
SCU registers	Yes	No	Yes	No
Processor interrupt interfaces				
Interrupt distributor				

- a. Halfword or doubleword accesses cause an abort to the requesting processor and the Fault Status Register shows this as a SLVERR.
- b. A word access with strobes not all set causes an abort to the requesting processor and the Fault Status Register shows this as a SLVERR.

The ACP cannot access any of the registers in this memory region.

[Table 2-9](#) shows register addresses for the Cortex-R7 MPCore processor relative to this base address.

**Table 2-9 Cortex-R7 MPCore private memory region**

Offset from PERIPHBASE[31:13]	Peripheral	Description
0x0000-0x00FC	SCU registers	<a href="#">SCU registers on page 9-5.</a>
0x0100-0x01FF	Interrupt controller interfaces	<a href="#">Interrupt controller on page 9-19.</a>
0x0200-0x02FF	Global timer	<a href="#">Global timer on page 9-35.</a>
0x0300-0x03FF	Reserved	Any access to this region causes a SLVERR abort exception.
0x0400-0x04FF		
0x0500-0x05FF		
0x0600-0x06FF	Private timers and watchdogs	<a href="#">Private timer and watchdog on page 9-29.</a>
0x0700-0x07FF	Reserved	Any access to this region causes a SLVERR abort exception.
0x0800-0x08FF		
0x0900-0x09FF		
0x0A00-0x0AFF		
0x0B00-0x0FFF		
0x1000-0x1FFF	Interrupt Distributor	<a href="#">Distributor register descriptions on page 9-20.</a>



# Chapter 3

## Programmers Model

This chapter describes the programmers model. It contains the following sections:

- *About the programmers model on page 3-2.*
- *The VFP extension on page 3-3.*
- *Multiprocessing extensions on page 3-4.*
- *Memory formats on page 3-5.*
- *Addresses in the Cortex-R7 MPCore processor on page 3-6.*

### 3.1 About the programmers model

The Cortex-R7 MPCore processor implements the ARMv7-R architecture. This includes:

- The 32-bit ARM instruction set.
- The Thumb instruction set that has both 16-bit and 32-bit instructions.
- *Vector Floating Point* (VFP) extensions.
- The Multiprocessing Extensions.

See the *ARM® Architecture Reference Manual ARMv7-A and ARMv7-R edition* for more information.

## 3.2 The VFP extension

The VFP extension performs single-precision and double-precision floating-point operations, and some half-precision operations.

There are build options to implement a single-precision FPU or a double-precision FPU. See [Chapter 5 Floating Point Unit Programmers Model](#) for implementation-specific information.

See the *ARM® Architecture Reference Manual ARMv7-A and ARMv7-R edition* for more information about the VFP extension.

### 3.3 Multiprocessing extensions

The multiprocessing extensions are a set of features that enhance multiprocessing functionality.

See [Chapter 4 \*System Control\*](#) and [Chapter 9 \*Multiprocessing\*](#) for implementation-specific information.

See the *ARM<sup>®</sup> Architecture Reference Manual ARMv7-A and ARMv7-R edition* for more information.

## 3.4 Memory formats

The Cortex-R7 MPCore processor views memory as a linear collection of bytes numbered in ascending order from zero. For example, bytes 0-3 hold the first stored word, and bytes 4-7 hold the second stored word.

The Cortex-R7 MPCore processor can store words in memory as either:

- Big-endian format.
- Little-endian format.

See the *ARM® Architecture Reference Manual ARMv7-A and ARMv7-R edition* for more information about big-endian and little-endian memory systems.

---

**Note**

---

Instructions are always treated as little-endian.

---

### 3.5 Addresses in the Cortex-R7 MPCore processor

All addresses are physical addresses. Address translation is not supported. See the *ARM® Architecture Reference Manual ARMv7-A and ARMv7-R edition* for a description of the default memory map.

The instruction and data address spaces are unified.

See the *ARM® Architecture Reference Manual ARMv7-A and ARMv7-R edition* for more information about memory addresses and access permissions. See [Fault handling on page 6-3](#) for implementation-specific information.

# Chapter 4

## System Control

This chapter describes the system control registers, their structure and operation, and how to use them. It contains the following sections:

- [About system control on page 4-2.](#)
- [Register summary on page 4-3.](#)
- [Register descriptions on page 4-17.](#)

## 4.1 About system control

The system control coprocessor, CP15, controls and provides status information for the functions implemented in the Cortex-R7 MPCore processor. There is one CP15 coprocessor for each processor in the Cortex-R7 MPCore processor. The main functions of the system control coprocessor are:

- Overall system control and configuration.
- MPU configuration and management.
- Cache configuration and management.
- TCM configuration and management.
- System performance monitoring.

In ARMv7-R the following instructions have instruction set equivalents:

- Instruction Synchronization Barrier.
- Data Synchronization Barrier.
- Data Memory Barrier.
- Wait for Interrupt.

The use of these registers is optional and deprecated.



## 4.2 Register summary

This section gives a summary of the CP15 system control registers. For more information on using the CP15 system control registers, see the *ARM® Architecture Reference Manual ARMv7-A and ARMv7-R edition*.

The system control coprocessor is a set of registers that you can write to and read from. Some of the registers permit more than one type of operation.

The following subsections describe the CP15 system control registers grouped by CRn order, and accessed by the MCR and MRC instructions in the order of CRn, Op1, CRm, Op2:

- [c0 registers on page 4-4.](#)
- [c1 registers on page 4-5.](#)
- [c5 registers on page 4-5.](#)
- [c6 registers on page 4-6.](#)
- [c7 registers on page 4-7.](#)
- [c9 registers on page 4-8.](#)
- [c13 registers on page 4-9.](#)
- [c15 registers on page 4-9.](#)

In addition to listing the CP15 system control registers by CRn ordering, the following subsections describe the CP15 system control registers by functional groups:

- [System identification, control, and configuration registers on page 4-11.](#)
- [Fault handling registers on page 4-12.](#)
- [MPU registers on page 4-13.](#)
- [Cache maintenance registers on page 4-14.](#)
- [Interface control and configuration registers on page 4-15.](#)
- [Performance monitor registers on page 4-15.](#)
- [Miscellaneous system control registers on page 4-16.](#)
- [Implementation-defined registers on page 4-16.](#)

[Table 4-1](#) describes the column headings that the CP15 register summary tables use throughout this section.

**Table 4-1 Column headings definition for CP15 register summary tables**

Column name	Description
CRn	Register number within the system control coprocessor
Op1	Opcode_1 value for the register
CRm	Operational register number within CRn
Op2	Opcode_2 value for the register
Name	Short form architectural, operation, or code name for the register
Reset	Reset value of register
Description	Cross-reference to register description

## 4.2.1 c0 registers

Table 4-2 shows the 32-bit wide CP15 system control registers when CRn is c0.

**Table 4-2 c0 register summary**

Op1	CRm	Op2	Name	Reset	Description
0	c0	0	MIDR	0x410FC171	<a href="#">Main ID Register on page 4-17</a>
		1	CTR	0x8333C003	Cache Type Register, see the <i>ARM® Architecture Reference Manual ARMv7-A and ARMv7-R edition</i>
		2	TCMTR	Implementation dependent <sup>a</sup>	TCM Type Register, see the <i>ARM® Architecture Reference Manual ARMv7-A and ARMv7-R edition</i>
		4	MPUIR	Implementation dependent <sup>b</sup>	<a href="#">MPU Type Register on page 4-17</a>
		5	MPIDR	Implementation dependent <sup>c</sup>	<a href="#">Multiprocessor Affinity Register on page 4-18</a>
		6	REVIDR	Implementation dependent	<a href="#">Revision ID Register on page 4-19</a>
	c1	0	ID_PFR0	0x00000131	Processor Feature Register 0, see the <i>ARM® Architecture Reference Manual ARMv7-A and ARMv7-R edition</i>
		1	ID_PFR1	0x00000001	Processor Feature Register 1, see the <i>ARM® Architecture Reference Manual ARMv7-A and ARMv7-R edition</i>
		2	ID_DFR0	0x00010404	Debug Feature Register, see the <i>ARM® Architecture Reference Manual ARMv7-A and ARMv7-R edition</i> .
		3	ID_AFR0	0x00000000	Auxiliary Feature Register 0, see the <i>ARM® Architecture Reference Manual ARMv7-A and ARMv7-R edition</i>
		4	ID_MMFR0	0x00110130	Memory Model Feature Register 0, see the <i>ARM® Architecture Reference Manual ARMv7-A and ARMv7-R edition</i>
		5	ID_MMFR1	0x00000000	Memory Model Feature Register 1, see the <i>ARM® Architecture Reference Manual ARMv7-A and ARMv7-R edition</i>
		6	ID_MMFR2	0x01200000	Memory Model Feature Register 2, see the <i>ARM® Architecture Reference Manual ARMv7-A and ARMv7-R edition</i>
		7	ID_MMFR3	0x00002111	Memory Model Feature Register 3, see the <i>ARM® Architecture Reference Manual ARMv7-A and ARMv7-R edition</i>
	c2	0	ID_ISAR0	0x02101111	Instruction Set Attributes Register 0, see the <i>ARM® Architecture Reference Manual ARMv7-A and ARMv7-R edition</i>
		1	ID_ISAR1	0x13112111	Instruction Set Attributes Register 1, see the <i>ARM® Architecture Reference Manual ARMv7-A and ARMv7-R edition</i>
		2	ID_ISAR2	0x21232141	Instruction Set Attributes Register 2, see the <i>ARM® Architecture Reference Manual ARMv7-A and ARMv7-R edition</i>
		3	ID_ISAR3	0x01112131	Instruction Set Attributes Register 3, see the <i>ARM® Architecture Reference Manual ARMv7-A and ARMv7-R edition</i>
		4	ID_ISAR4	0x00010142	Instruction Set Attributes Register 4, see the <i>ARM® Architecture Reference Manual ARMv7-A and ARMv7-R edition</i>

**Table 4-2 c0 register summary (continued)**

Op1	CRm	Op2	Name	Reset	Description
1	c0	0	CCSIDR	UNK	<i>Cache Size ID Register on page 4-20</i>
		1	CLIDR	Implementation dependent <sup>d</sup>	<i>Cache Level ID Register on page 4-21</i>
		7	AIDR	0x00000000	<i>Auxiliary ID Register on page 4-22</i>
2	c0	0	CSSELR	Implementation dependent	<i>Cache Size Selection Register on page 4-22</i>

- a. If TCMs are implemented 0x80010001.  
If TCMs are not implemented 0x00000000.
- b. For 12 MPU regions 0x00000c00.  
For 16 MPU regions 0x00001000.
- c. Dependent on external signal **CLUSTERID** and the number of configured processors in the Cortex-R7 MPCore processor.
- d. If cache present 0x09200003.  
If cache not present 0x00000000.

#### 4.2.2 c1 registers

Table 4-3 shows the 32-bit wide CP15 system control registers when CRn is c1.

**Table 4-3 c1 register summary**

Op1	CRm	Op2	Name	Reset	Description
0	c0	0	SCTLR	UNK	<i>System Control Register on page 4-23</i>
		1	ACTLR	0x00000000	<i>Auxiliary Control Register on page 4-25</i>
		2	CPACR	0xC0000000	<i>Coprocessor Access Control Register on page 4-27</i>

#### 4.2.3 c5 registers

Table 4-4 shows the 32-bit wide CP15 system control registers when CRn is c5.

**Table 4-4 c5 register summary**

Op1	CRm	Op2	Name	Reset	Description
0	c0	0	DFSR	-	Data Fault Status Register, see the <i>ARM® Architecture Reference Manual ARMv7-A and ARMv7-R edition</i>
		1	IFSR	-	Instruction Fault Status Register, see the <i>ARM® Architecture Reference Manual ARMv7-A and ARMv7-R edition</i>

#### 4.2.4 c6 registers

Table 4-5 shows the 32-bit wide CP15 system control registers when CRn is c6.

**Table 4-5 c6 register summary**

Op1	CRm	Op2	Name	Reset	Description
0	c0	0	DFAR	-	Data Fault Address Register, see the <i>ARM® Architecture Reference Manual ARMv7-A and ARMv7-R edition</i>
		2	IFAR	-	Instruction Fault Address Register, see the <i>ARM® Architecture Reference Manual ARMv7-A and ARMv7-R edition</i>
	c1	0	DRBAR	UNK	<a href="#">MPU Region Base Address Registers on page 4-29</a>
		2	DRSR	0x00000000	<a href="#">MPU Region Size and Enable Registers on page 4-30</a>
		4	DRACR	UNK	<a href="#">MPU Region Access Control Registers on page 4-31</a>
	c2	0	RGNR	UNK	<a href="#">MPU Memory Region Number Registers on page 4-32</a>

## 4.2.5 c7 registers

Table 4-6 shows the 32-bit wide CP15 system control registers when CRn is c7.

**Table 4-6 c7 register summary**

Op1	CRm	Op2	Name	Reset <sup>a</sup>	Description
0	c0	4	NOP	-	No operation, see the <i>ARM® Architecture Reference Manual ARMv7-A and ARMv7-R edition</i> .
	c1	0	ICIALUIS	-	Invalidate all instruction caches to PoU Inner Shareable, see the <i>ARM® Architecture Reference Manual ARMv7-A and ARMv7-R edition</i>
		6	BPIALLIS	-	Invalidate entire branch predictor array Inner Shareable, see the <i>ARM® Architecture Reference Manual ARMv7-A and ARMv7-R edition</i>
	c5	0	ICIALLU	-	Invalidate entire instruction cache, see the <i>ARM® Architecture Reference Manual ARMv7-A and ARMv7-R edition</i>
		1	ICIMVAU	-	Invalidate instruction cache by VA to PoU, see the <i>ARM® Architecture Reference Manual ARMv7-A and ARMv7-R edition</i>
		4	CP15ISB	-	Instruction Synchronization Barrier operation, see the <i>ARM® Architecture Reference Manual ARMv7-A and ARMv7-R edition</i>
		6	BPIALL	-	Invalidate entire branch predictor array, see the <i>ARM® Architecture Reference Manual ARMv7-A and ARMv7-R edition</i>
		7	BPIMVA	-	Invalidate MVA from branch predictors, see the <i>ARM® Architecture Reference Manual ARMv7-A and ARMv7-R edition</i>
	c6	1	DCIMVAC	-	Invalidate data cache line by VA to PoC, see the <i>ARM® Architecture Reference Manual ARMv7-A and ARMv7-R edition</i>
		2	DCISW	-	Invalidate data cache line by Set/Way, see the <i>ARM® Architecture Reference Manual ARMv7-A and ARMv7-R edition</i>
	c10	1	DCCMVAC	-	Clean data cache line to PoC by VA, see the <i>ARM® Architecture Reference Manual ARMv7-A and ARMv7-R edition</i>
		2	DCCSW	-	Clean data cache line by Set/Way, see the <i>ARM® Architecture Reference Manual ARMv7-A and ARMv7-R edition</i>
		4	CP15DSB	-	Data Synchronization Barrier operation, see the <i>ARM® Architecture Reference Manual ARMv7-A and ARMv7-R edition</i>
		5	CP15DMB	-	Data Memory Barrier operation, see the <i>ARM® Architecture Reference Manual ARMv7-A and ARMv7-R edition</i>
0	c11	1	DCCMVAU	-	Clean data or unified cache line by VA to PoU, see the <i>ARM® Architecture Reference Manual ARMv7-A and ARMv7-R edition</i>
	c14	1	DCCIMVAC	-	Clean and invalidate data cache line by VA to PoC, see the <i>ARM® Architecture Reference Manual ARMv7-A and ARMv7-R edition</i>
		2	DCCISW	-	Clean and invalidate data cache line by Set/Way, see the <i>ARM® Architecture Reference Manual ARMv7-A and ARMv7-R edition</i>

a. These registers do not have a reset value because they are write-only.

## 4.2.6 c9 registers

Table 4-7 shows the 32-bit wide CP15 system control registers when CRn is c9.

**Table 4-7 c9 register summary**

Op1	CRm	Op2	Name	Reset	Description
0	c1	0	DTCMRR	UNK	<i>DTCM Region Register on page 4-33</i>
		1	ITCMRR	UNK	<i>ITCM Region Register on page 4-34</i>
	c12	0	PMCR	0x41174000	Performance Monitor Control Register, see the <i>ARM® Architecture Reference Manual ARMv7-A and ARMv7-R edition</i>
		1	PMCNTENSET	0x00000000	Count Enable Set Register, see the <i>ARM® Architecture Reference Manual ARMv7-A and ARMv7-R edition</i>
		2	PMCNTENCLR	0x00000000	Count Enable Clear Register, see the <i>ARM® Architecture Reference Manual ARMv7-A and ARMv7-R edition</i>
		3	PMOVSr	0x00000000	Overflow Flag Status Register, see the <i>ARM® Architecture Reference Manual ARMv7-A and ARMv7-R edition</i>
		4	PMSWINC	UNK	Software Increment Register, see the <i>ARM® Architecture Reference Manual ARMv7-A and ARMv7-R edition</i>
		5	PMSELR	0x00000000	Event Counter Selection Register, see the <i>ARM® Architecture Reference Manual ARMv7-A and ARMv7-R edition</i>
	c13	0	PMCCNTR	UNK	Cycle Count Register, see the <i>ARM® Architecture Reference Manual ARMv7-A and ARMv7-R edition</i> .
		1	PMXEVTYPER	UNK	Event Selection Register, see the <i>ARM® Architecture Reference Manual ARMv7-A and ARMv7-R edition</i>
		2	PMXEVCNTR	UNK	Performance Monitor Count Registers, see the <i>ARM® Architecture Reference Manual ARMv7-A and ARMv7-R edition</i>
	c14	0	PMUSERENR	0x00000000	User Enable Register, see the <i>ARM® Architecture Reference Manual ARMv7-A and ARMv7-R edition</i>
		1	PMINTENSET	0x00000000	Interrupt Enable Set Register, see the <i>ARM® Architecture Reference Manual ARMv7-A and ARMv7-R edition</i>
		2	PMINTENCLR	0x00000000	Interrupt Enable Clear Register, see the <i>ARM® Architecture Reference Manual ARMv7-A and ARMv7-R edition</i>

### 4.2.7 c13 registers

Table 4-8 shows the 32-bit wide CP15 system control registers when CRn is c13.

**Table 4-8 c13 register summary**

Op1	CRm	Op2	Name	Reset	Description
0	c0	1	CONTEXTIDR	UNK	Context ID Register, see the <i>ARM® Architecture Reference Manual ARMv7-A and ARMv7-R edition</i>
		2	TPIDRURW	UNK	User Read/Write Software Thread Register, see the <i>ARM® Architecture Reference Manual ARMv7-A and ARMv7-R edition</i>
		3	TPIDRURO	UNK	User Read Only Software Thread Register, see the <i>ARM® Architecture Reference Manual ARMv7-A and ARMv7-R edition</i>
		4	TPIDRPRW	UNK	Privileged Only Software Thread Register, see the <i>ARM® Architecture Reference Manual ARMv7-A and ARMv7-R edition</i>

### 4.2.8 c15 registers

Table 4-9 shows the 32-bit wide CP15 system control registers when CRn is c15.

**Table 4-9 c15 register summary**

Op1	CRm	Op2	Name	Reset	Description
0	c0	0	PCR	0x00000020	<a href="#">Power Control Register on page 4-35</a>
		c1	CTDOR	UNK	<a href="#">Cache and TCM Debug Operation Register on page 4-36</a>
	c1	1	RADRLO	UNK	<a href="#">RAM Access Data Registers on page 4-38, bits[31:0]</a>
		2	RADRH1	UNK	<a href="#">RAM Access Data Registers on page 4-38, bits[63:32]</a>
		3	RAECCR <sup>a</sup>	UNK	<a href="#">RAM Access ECC Register on page 4-39</a>
	c2	0	D_ECC_ENTRY_0 <sup>a</sup>	UNK	<a href="#">ECC Error Registers on page 4-39</a>
		1	D_ECC_ENTRY_1 <sup>a</sup>	UNK	
		2	D_ECC_ENTRY_2 <sup>a</sup>	UNK	
	c3	0	I_ECC_ENTRY_0 <sup>a</sup>	UNK	
		1	I_ECC_ENTRY_1 <sup>a</sup>	UNK	
		2	I_ECC_ENTRY_2 <sup>a</sup>	UNK	
	c4	0	DTCM_ECC_ENTRY <sup>b</sup>	UNK	
	c5	0	ITCM_ECC_ENTRY <sup>b</sup>	UNK	
4	c0	0	CBAR	UNK	<a href="#">Configuration Base Address Register on page 4-42</a>

a. Only present if ECC is present, otherwise RAZ/WI.

b. Only present if ECC and TCM are present, otherwise RAZ/WI.

#### 4.2.9 System identification, control, and configuration register

[Table 4-10 on page 4-11](#) shows the system identification, control, and configuration registers.



Table 4-10 System identification, control, and configuration registers

Name	CRn	Op1	CRm	Op2	Reset	Description
MIDR	c0	0	c0	0	0x410FC171	<a href="#">Main ID Register on page 4-17</a>
CTR				1	0x8333C003	Cache Type Register, see the <i>ARM® Architecture Reference Manual ARMv7-A and ARMv7-R edition</i>
TCMTR				2	Implementation dependent <sup>a</sup>	TCM Type Register, see the <i>ARM® Architecture Reference Manual ARMv7-A and ARMv7-R edition</i>
MPUIR				4	Implementation dependent <sup>b</sup>	<a href="#">MPU Type Register on page 4-17</a>
MPIDR				5	Implementation dependent <sup>c</sup>	<a href="#">Multiprocessor Affinity Register on page 4-18</a>
REVIDR				6	Implementation dependent	<a href="#">Revision ID Register on page 4-19</a>
ID_PFR0			c1	0	0x00000131	Processor Feature Register 0, see the <i>ARM® Architecture Reference Manual ARMv7-A and ARMv7-R edition</i>
ID_PFR1				1	0x00000001	Processor Feature Register 1, see the <i>ARM® Architecture Reference Manual ARMv7-A and ARMv7-R edition</i>
ID_DFR0				2	0x00010404	Debug Feature Register, see the <i>ARM® Architecture Reference Manual ARMv7-A and ARMv7-R edition</i>
ID_AFR0				3	0x00000000	Auxiliary Feature Register 0, see the <i>ARM® Architecture Reference Manual ARMv7-A and ARMv7-R edition</i>
ID_MMFR0				4	0x00110130	Memory Model Feature Register 0, see the <i>ARM® Architecture Reference Manual ARMv7-A and ARMv7-R edition</i>
ID_MMFR1				5	0x00000000	Memory Model Feature Register 1, see the <i>ARM® Architecture Reference Manual ARMv7-A and ARMv7-R edition</i>
ID_MMFR2				6	0x01200000	Memory Model Feature Register 2, see the <i>ARM Architecture Reference Manual</i>
ID_MMFR3				7	0x00002111	Memory Model Feature Register 3, see the <i>ARM® Architecture Reference Manual ARMv7-A and ARMv7-R edition</i>
ID_ISAR0			c2	0	0x02101111	Instruction Set Attributes Register 0, see the <i>ARM® Architecture Reference Manual ARMv7-A and ARMv7-R edition</i>
ID_ISAR1				1	0x13112111	Instruction Set Attributes Register 1, see the <i>ARM® Architecture Reference Manual ARMv7-A and ARMv7-R edition</i>
ID_ISAR2				2	0x21232141	Instruction Set Attributes Register 2, see the <i>ARM® Architecture Reference Manual ARMv7-A and ARMv7-R edition</i>
ID_ISAR3				3	0x01112131	Instruction Set Attributes Register 3, see the <i>ARM® Architecture Reference Manual ARMv7-A and ARMv7-R edition</i>

Table 4-10 System identification, control, and configuration registers (continued)

Name	CRn	Op1	CRm	Op2	Reset	Description
CCSIDR	c0	1	c0	0	UNK <sup>d</sup>	<a href="#">Cache Size ID Register on page 4-20</a>
CLIDR				1	Implementation dependent <sup>e</sup>	<a href="#">Cache Level ID Register on page 4-21</a>
AIDR				7	0x00000000	<a href="#">Auxiliary ID Register on page 4-22</a>
CSSELR		2	c0	0	Implementation dependent	<a href="#">Cache Size Selection Register on page 4-22</a>
SCTLR	c1	0	c0	0	-	<a href="#">System Control Register on page 4-23</a>
ACTLR				1	0x00000000	<a href="#">Auxiliary Control Register on page 4-25</a>
CPACR				2	0xC0000000	<a href="#">Coprocessor Access Control Register on page 4-27</a>

- a. If TCMs are implemented 0x80010001.  
If TCMs are not implemented 0x00000000.
- b. For 12 MPU regions 0x0000c00.  
For 16 MPU regions 0x00001000.
- c. Dependent on external signal **CLUSTERID** and the number of configured processors in the Cortex-R7 MPCore processor.
- d. Dependent on cache sizes and whether cache is on or off.
- e. If cache present 0x09200003.  
If cache not present 0x00000000.

#### 4.2.10 Fault handling registers

Table 4-11 shows the fault handling registers.

Table 4-11 Fault handling registers

Name	CRn	Op1	CRm	Op2	Reset	Description
DFSR	c5	0	c0	0	-	Data Fault Status Register, see the <i>ARM® Architecture Reference Manual ARMv7-A and ARMv7-R edition</i>
IFSR				1	-	Instruction Fault Status Register, see the <i>ARM® Architecture Reference Manual ARMv7-A and ARMv7-R edition</i>
DFAR	c6	0	c0	0	-	Data Fault Address Register, see the <i>ARM® Architecture Reference Manual ARMv7-A and ARMv7-R edition</i>
IFAR				2	-	Instruction Fault Address Register, see the <i>ARM® Architecture Reference Manual ARMv7-A and ARMv7-R edition</i>

### 4.2.11 MPU registers

Table 4-12 shows the *Memory Protection Unit* (MPU) registers.

**Table 4-12 MPU registers**

Name	CRn	Op1	CRm	Op2	Reset	Description
DRBAR	c6	0	c1	0	UNK	<a href="#">MPU Region Base Address Registers on page 4-29</a>
DRSR				2	0x00000000	<a href="#">MPU Region Size and Enable Registers on page 4-30</a>
DRACR				4	UNK	<a href="#">MPU Region Access Control Registers on page 4-31</a>
RGNR			c2	0	UNK	<a href="#">MPU Memory Region Number Registers on page 4-32</a>

## 4.2.12 Cache maintenance registers

Table 4-13 shows the cache maintenance registers.

**Table 4-13 Cache maintenance registers**

Name	CRn	Op1	CRm	Op2	Reset	Description
NOP	c7	0	c0	4	-	No operation, see the <i>ARM® Architecture Reference Manual ARMv7-A and ARMv7-R edition</i>
ICIALUIS			c1	0	-	Invalidate all instruction caches to PoU Inner Shareable, see the <i>ARM® Architecture Reference Manual ARMv7-A and ARMv7-R edition</i>
BPIALLIS				6	-	Invalidate entire branch predictor array Inner Shareable, see the <i>ARM® Architecture Reference Manual ARMv7-A and ARMv7-R edition</i>
ICIALLU			c5	0	-	Invalidate entire instruction cache, see the <i>ARM® Architecture Reference Manual ARMv7-A and ARMv7-R edition</i>
ICIMVAU				1	-	Invalidate instruction cache by VA to PoU, see the <i>ARM® Architecture Reference Manual ARMv7-A and ARMv7-R edition</i>
CP15ISB				4	-	Instruction Synchronization Barrier operation, see the <i>ARM® Architecture Reference Manual ARMv7-A and ARMv7-R edition</i>
BPIALL				6	-	Invalidate entire branch predictor array, see the <i>ARM® Architecture Reference Manual ARMv7-A and ARMv7-R edition</i>
BPIMVA					-	Invalidate MVA from branch predictors, see the <i>ARM® Architecture Reference Manual ARMv7-A and ARMv7-R edition</i>
DCIMVAC			c6	1	-	Invalidate data cache line by VA to PoC, see the <i>ARM® Architecture Reference Manual ARMv7-A and ARMv7-R edition</i>
DCISW				2	-	Invalidate data cache line by Set/Way, see the <i>ARM® Architecture Reference Manual ARMv7-A and ARMv7-R edition</i>
DCCMVAC			c10	1	-	Clean data cache line to PoC by VA, see the <i>ARM® Architecture Reference Manual ARMv7-A and ARMv7-R edition</i>
DCCSW				2	-	Clean data cache line by Set/Way. see the <i>ARM® Architecture Reference Manual ARMv7-A and ARMv7-R edition</i>
CP15DSB			c10	4	-	Data Synchronization Barrier operation, see the <i>ARM® Architecture Reference Manual ARMv7-A and ARMv7-R edition</i>
CP15DMB				5	-	Data Memory Barrier operation, see the <i>ARM® Architecture Reference Manual ARMv7-A and ARMv7-R edition</i>
DCCMVAU			c11	1	-	Clean data or unified cache line by VA to PoU, see the <i>ARM® Architecture Reference Manual ARMv7-A and ARMv7-R edition</i>
DCCIMVAC			c14	1	-	Clean and invalidate data cache line by VA to PoC, see the <i>ARM® Architecture Reference Manual ARMv7-A and ARMv7-R edition</i>
DCCISW				2	-	Clean and invalidate data cache line by Set/Way, see the <i>ARM® Architecture Reference Manual ARMv7-A and ARMv7-R edition</i>

### 4.2.13 Interface control and configuration registers

Table 4-14 shows the interface control and configuration registers.

**Table 4-14 Interface control and configuration registers**

Name	CRn	Op1	CRm	Op2	Reset	Description
DTCMRR	c9	0	c1	0	UNK	<a href="#">DTCM Region Register on page 4-33</a>
ITCMRR				1	UNK	<a href="#">ITCM Region Register on page 4-34</a>

### 4.2.14 Performance monitor registers

Table 4-15 shows the performance monitor registers.

**Table 4-15 Performance monitor registers**

Name	CRn	Op1	CRm	Op2	Reset	Description
PMCR	c9	0	c12	0	0x41104000	Performance Monitor Control Register, see the <i>ARM® Architecture Reference Manual ARMv7-A and ARMv7-R edition</i>
PMCNTENSET				1	0x00000000	Count Enable Set Register, see the <i>ARM® Architecture Reference Manual ARMv7-A and ARMv7-R edition</i>
PMCNTENCLR				2	0x00000000	Count Enable Clear Register, see the <i>ARM® Architecture Reference Manual ARMv7-A and ARMv7-R edition</i>
PMOVSr				3	0x00000000	Overflow Flag Status Register, see the <i>ARM® Architecture Reference Manual ARMv7-A and ARMv7-R edition</i>
PMSWINC				4	UNK	Software Increment Register, see the <i>ARM® Architecture Reference Manual ARMv7-A and ARMv7-R edition</i>
PMSELR				5	0x00000000	Event Counter Selection Register, see the <i>ARM® Architecture Reference Manual ARMv7-A and ARMv7-R edition</i>
PMCCNTR			c13	0	UNK	Cycle Count Register, see the <i>ARM® Architecture Reference Manual ARMv7-A and ARMv7-R edition</i>
PMXEVTYPER				1	UNK	Event Selection Register, see the <i>ARM® Architecture Reference Manual ARMv7-A and ARMv7-R edition</i>
PMXEVCNTR	c9	0	c13	2	UNK	Performance Monitor Count Registers, see the <i>ARM® Architecture Reference Manual ARMv7-A and ARMv7-R edition</i>
PMUSERENR			c14	0	0x00000000	User Enable Register, see the <i>ARM® Architecture Reference Manual ARMv7-A and ARMv7-R edition</i>
PMINTENSET				1	0x00000000	Interrupt Enable Set Register, see the <i>ARM® Architecture Reference Manual ARMv7-A and ARMv7-R edition</i>
PMINTENCLR				2	0x00000000	Interrupt Enable Clear Register, see the <i>ARM® Architecture Reference Manual ARMv7-A and ARMv7-R edition</i>

#### 4.2.15 Miscellaneous system control registers

Table 4-16 shows the miscellaneous system control registers.

**Table 4-16 Miscellaneous system control registers**

Name	CRn	Op1	CRm	Op2	Reset	Description
CONTEXTIDR	c13	0	c0	1	UNK	Context ID Register, see the <i>ARM® Architecture Reference Manual ARMv7-A and ARMv7-R edition</i>
TPIDRURW				2	UNK	User Read/Write Software Thread Register, see the <i>ARM® Architecture Reference Manual ARMv7-A and ARMv7-R edition</i>
TPIDRURO				3	UNK	User Read Only Software Thread Register, see the <i>ARM® Architecture Reference Manual ARMv7-A and ARMv7-R edition</i>
TPIDRPRW				4	UNK	Privileged Only Software Thread Register, see the <i>ARM® Architecture Reference Manual ARMv7-A and ARMv7-R edition</i>

#### 4.2.16 Implementation-defined registers

Table 4-17 shows the implementation-defined registers.

**Table 4-17 Implementation-defined registers**

Name	CRn	Op1	CRm	Op2	Reset	Description
PCR	c15	0	c0	0	0x00000020	<a href="#">Power Control Register on page 4-35</a>
CTDOR			c1	0	UNK	<a href="#">Cache and TCM Debug Operation Register on page 4-36</a>
RADRLO				1	UNK	<a href="#">RAM Access Data Registers on page 4-38, bits[31:0]</a>
RADRHI				2	UNK	<a href="#">RAM Access Data Registers on page 4-38, bits[63:32]</a>
RAECCR <sup>a</sup>				3	UNK	<a href="#">RAM Access ECC Register on page 4-39</a>
D_ECC_ENTRY_0 <sup>a</sup>			c2	0	UNK	<a href="#">ECC Error Registers on page 4-39</a>
D_ECC_ENTRY_1 <sup>a</sup>				1	UNK	
D_ECC_ENTRY_2 <sup>a</sup>				2	UNK	
I_ECC_ENTRY_0 <sup>a</sup>			c3	0	UNK	
I_ECC_ENTRY_1 <sup>a</sup>				1	UNK	
I_ECC_ENTRY_2 <sup>a</sup>				2	UNK	
DTCM_ECC_ENTRY <sup>b</sup>			c4	0	UNK	
ITCM_ECC_ENTRY <sup>b</sup>			c5	0	UNK	
CBAR		4	c0	0	UNK	<a href="#">Configuration Base Address Register on page 4-42</a>

a. Only present if ECC is present, otherwise RAZ/WI.

b. Only present if ECC and TCM are present, otherwise RAZ/WI.

## 4.3 Register descriptions

This section describes the CP15 system control registers in coprocessor register number order. [Table 4-2 on page 4-4](#) to [Table 4-9 on page 4-9](#) provide cross references to individual registers

### 4.3.1 Main ID Register

The MIDR characteristics are:

**Purpose** Provides identification information for the Cortex-R7 MPCore processor, including an implementer code for the device and a device ID number.

**Usage constraints**

The MIDR is:

- Only accessible in privileged mode.
- A read-only register.

**Configurations** Available in all configurations.

**Attributes** See the register summary in [Table 4-2 on page 4-4](#).

[Figure 4-1](#) shows the MIDR bit assignments.

31	24	23	20	19	16	15					4	3	0
Implementer				Variant		Architecture		Primary part number				Revision	

**Figure 4-1 MIDR bit assignments**

[Table 4-18](#) shows the MIDR bit assignments.

**Table 4-18 MIDR bit assignments**

Bits	Name	Function
[31:24]	Implementer	Indicates the implementer code.
[23:20]	Variant	Indicates the variant number of the processor. This is the major revision number <i>n</i> of the <i>rnpn</i> revision status.
[19:16]	Architecture	Indicates the architecture code.
[15:4]	Primary part number	Indicates the primary part number.
[3:0]	Revision	Indicates the revision number of the processor. This is the minor revision number <i>n</i> of the <i>rnpn</i> revision status.

To access the MIDR, read the CP15 register with:

MRC p15, 0, <Rd>, c0, c0, 0; Read Main ID Register

### 4.3.2 MPU Type Register

The MPUIR characteristics are:

**Purpose** Indicates:

- The number of MPU regions, 12 or 16.
- The type of MPU regions, unified or separate.

- Usage constraints

The MPUIR is:
  - Only accessible in privileged mode.
  - A read-only register.
- Configurations

Available in all configurations.
- Attributes

See the register summary in [Table 4-2 on page 4-4](#).

Figure 4-2 shows the MPUIR bit assignments.

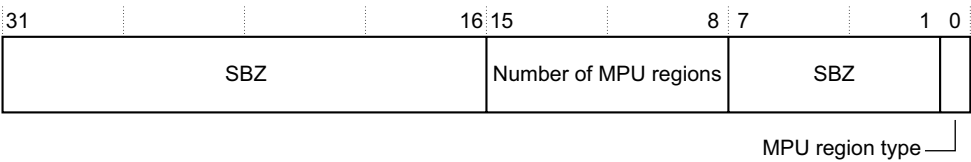


Figure 4-2 MPUIR bit assignments

Table 4-19 shows the MPUIR bit assignments.

Table 4-19 MPUIR bit assignments

Bits	Name	Function
[31:16]	Reserved	SBZ
[15:8]	Number of MPU regions	Indicates the number of regions: 00b00010000 16 regions. 00b00001100 12 regions.
[7:1]	Reserved	SBZ
[0]	MPU region type	Specifies the type of MPU regions, unified or separate, in the processor. Always set to 0 because the Cortex-R7 MPCore processor has unified memory regions. See <a href="#">Addresses in the Cortex-R7 MPCore processor on page 3-6</a> .

To access the MPUIR, read the CP15 register with:  
MRC p15,0,<Rt>,c0,c0,4 ; Read CP15 MPU Type Register

4.3.3 Multiprocessor Affinity Register

- The MPIDR characteristics are:
- Purpose

Provides an additional processor identification mechanism for scheduling purposes in a multiprocessor system.
- Usage constraints

The MPIDR is only accessible in privileged mode.
- Configurations

Available in all configurations. The value of the U bit, bit [30], indicates a multiprocessor or a uniprocessor configuration.
- Attributes

See the register summary in [Table 4-2 on page 4-4](#).

Figure 4-3 on page 4-19 shows the MPIDR bit assignments.



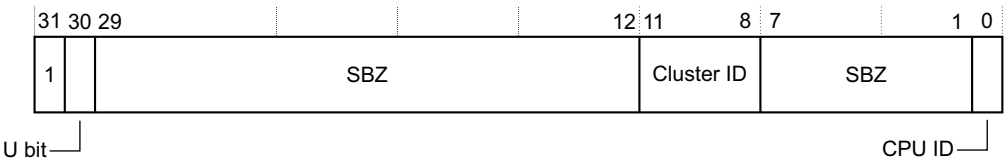


Figure 4-3 MPIDR bit assignments

Table 4-20 shows the MPIDR bit assignments.

Table 4-20 MPIDR bit assignments

Bits	Name	Function
[31]	-	Indicates the register uses the new multiprocessor format. This is always 1.
[30]	U bit	Multiprocessing Extensions: Set to 0 Processor is part of a multiprocessor cluster.
[29:12]	Reserved	SBZ.
[11:8]	Cluster ID	Value read in <b>CLUSTERID</b> configuration inputs. It identifies a Cortex-R7 processor in a system with more than one Cortex-R7 processor present.
[7:1]	Reserved	SBZ.
[0]	CPU ID	Indicates the processor number in the Cortex-R7 MPCore configuration: 0x0 Processor 0. 0x1 Processor 1.

To access the MPIDR, read the CP15 register with:

MRC p15,0,<Rd>,c0,c0,5; read Multiprocessor ID register

4.3.4 Revision ID Register

The REVIDR characteristics are:

- Purpose** Provides implementation-specific minor revision information that can only be interpreted in conjunction with the MIDR.
- Usage constraints** The REVIDR is only accessible in privileged mode.
- Configurations** Available in all configurations.
- Attributes** See the register summary in Table 4-2 on page 4-4.

Figure 4-4 shows the REVIDR bit assignments.

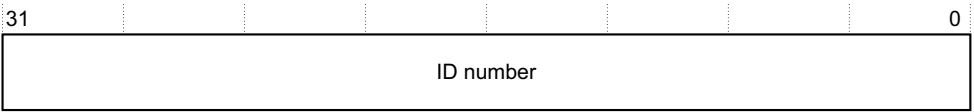


Figure 4-4 REVIDR bit assignments

Table 4-21 shows the REVIDR bit assignments.

**Table 4-21 REVIDR bit assignments**

Bits	Name	Function
[31:0]	ID number	Implementation-specific revision information. The reset value is determined by the specific Cortex-R7 MPCore processor implementation.

To access the REVIDR, read the CP15 register with:

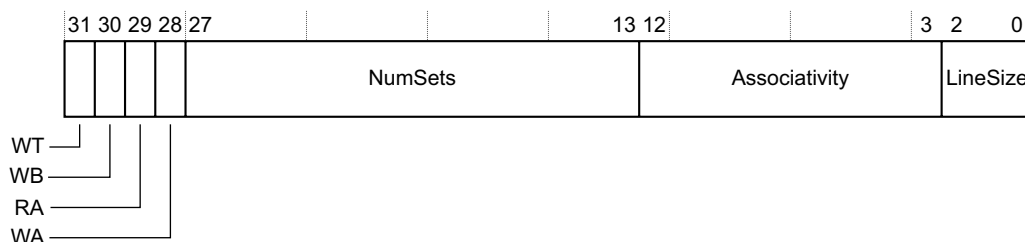
MRC p15,0,<Rd>,c0,c0,6; read Revision ID register

### 4.3.5 Cache Size ID Register

The CCSIDR characteristics are:

<b>Purpose</b>	Provides information about the architecture of the caches selected by CSSELR.
<b>Usage constraints</b>	The CCSIDR is only accessible in privileged mode.
<b>Configurations</b>	Available in configurations with caches implemented. If caches are not implemented, the value of this register is unknown.
<b>Attributes</b>	See the register summary in Table 4-2 on page 4-4.

Figure 4-5 shows the CCSIDR bit assignments.



**Figure 4-5 CCSIDR bit assignments**

Table 4-22 shows the CCSIDR bit assignments.

**Table 4-22 CCSIDR bit assignments**

Bits	Name	Function
[31]	WT	Indicates support available for Write-Through: Set to 0 Write-Through support not available.
[30]	WB	Indicates support available for Write-Back: 0 Write-Back support not available. 1 Write-Back support available.
[29]	RA	Indicates support available for read allocation: 0 Read allocation support not available. 1 Read allocation support available.
[28]	WA	Indicates support available for write allocation: 0 Write allocation support not available. 1 Write allocation support available.

Table 4-22 CCSIDR bit assignments (continued)

Bits	Name	Function
[27:13]	NumSets	Indicates number of sets:
		0x1F      4KB cache size.
		0x3F      8KB cache size.
		0x7F      16KB cache size.
		0xFF      32KB cache size.
		0x1FF      64KB cache size.
[12:3]	Associativity	Indicates number of ways:
		0b000000011      Four ways.
[2:0]	LineSize	Indicates number of words:
		0b001      Eight words per line.

To access the CCSIDR, read the CP15 register with:

MRC p15, 1, <Rd>, c0, c0, 0; Read current Cache Size Identification Register

If the CSSELR reads the instruction cache values and caches are implemented, bits[31:28] are 0b0010.

If the CSSELR reads the data cache values and caches are implemented, bits[31:28] are 0b0111. See [Cache Size Selection Register on page 4-22](#).

### 4.3.6 Cache Level ID Register

The CLIDR characteristics are:

**Purpose** Indicates the cache levels that are implemented in the Cortex-R7 MPCore processor and under the control of the System Control Coprocessor. If caches are not implemented, this register value is 0.

**Usage constraints** The CLIDR is:

- Only accessible in privileged mode.
- A read-only register.

**Configurations** Available in all configurations.

**Attributes** See the register summary in [Table 4-2 on page 4-4](#).

Figure 4-6 shows the CLIDR bit assignments.

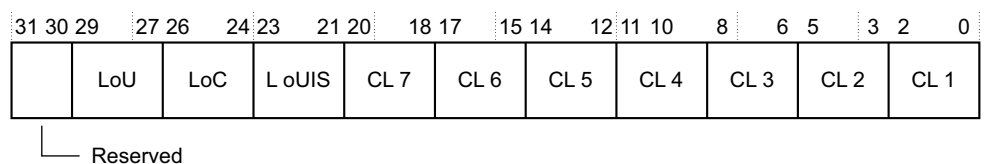


Figure 4-6 CLIDR bit assignments

Table 4-23 shows the CLIDR bit assignments.

**Table 4-23 CLIDR bit assignments**

Bits	Name	Function
[31:30]	-	UNP or SBZ
[29:27]	LoU	0b001 Level of unification.
[26:24]	LoC	0b001 Level of coherency.
[23:21]	LoUIS	0b001 Level of Unification Inner Shareable.
[20:18]	CL 7	0b000 No cache at CL 7.
[17:15]	CL 6	0b000 No cache at CL 6.
[14:12]	CL 5	0b000 No cache at CL 5.
[11:9]	CL 4	0b000 No cache at CL 4.
[8:6]	CL 3	0b000 No cache at CL 3.
[5:3]	CL 2	0b000. No unified cache at CL 2
[2:0]	CL 1	0b000 Caches not implemented. 0b011 Separate instruction and data caches at CL 1.

To access the CLIDR, read the CP15 register with:

MRC p15, 1,<Rd>, c0, c0, 1; Read CLIDR

### 4.3.7 Auxiliary ID Register

Not implemented.

### 4.3.8 Cache Size Selection Register

The CSSELR characteristics are:

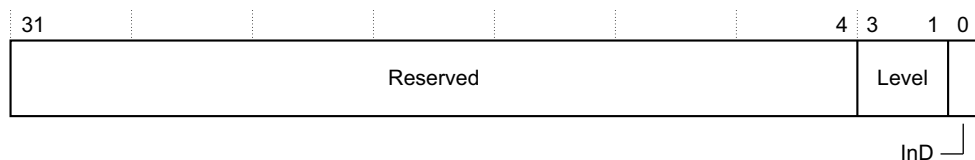
**Purpose** Selects the current CCSIDR.

**Usage constraints** The CSSELR is only accessible in privileged mode.

**Configurations** Available in all configurations.

**Attributes** See the register summary in Table 4-2 on page 4-4.

Figure 4-7 shows the CSSELR bit assignments.



**Figure 4-7 CSSELR bit assignments**

Table 4-24 shows the CSSELR bit assignments.

**Table 4-24 CSSELR bit assignments**

Bits	Name	Function
[31:4]	-	UNP or SBZ.
[3:1]	Level	Cache level selected, RAZ/WI. There is only one level of cache in the Cortex-R7 MPCore processor so the value for this field is 0b000.
[0]	InD	1 Instruction cache. 0 Data cache.

To access the CSSELR, read the CP15 register with:

```
MRC p15, 2, <Rd>, c0, c0, 0; Read CSSELR
MCR p15, 2, <Rd>, c0, c0, 0; Write CSSELR
```

### 4.3.9 System Control Register

The SCTLR characteristics are:

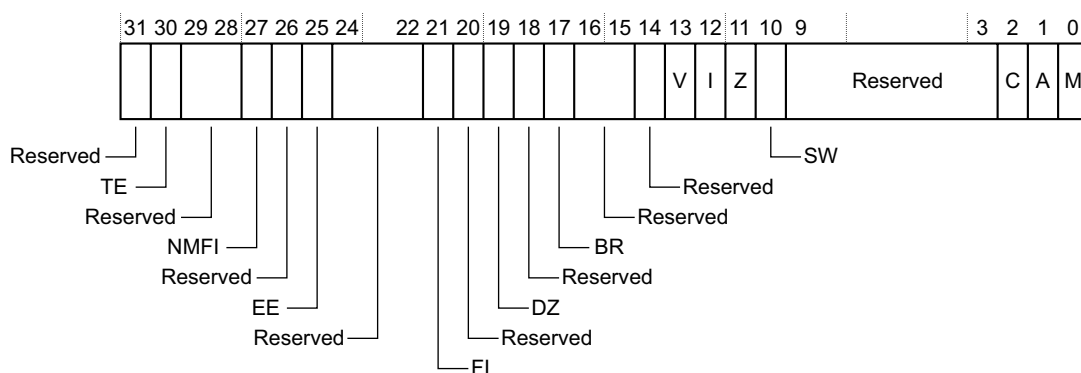
- Purpose** Provides control and configuration of:
- Memory alignment and endianness.
  - Memory protection and fault behavior.
  - MPU and cache enables.
  - Interrupts and behavior of interrupt latency.
  - Location for exception vectors.
  - Program flow prediction.

**Usage constraints** The SCTLR is only accessible in privileged mode.

**Configurations** Available in all configurations.

**Attributes** See the register summary in [Table 4-3 on page 4-5](#).

Figure 4-8 shows the SCTLR bit assignments.



**Figure 4-8 SCTLR bit assignments**

Table 4-25 shows the SCTLR bit assignments.

**Table 4-25 SCTLR bit assignments**

Bits	Name	Access	Function
[31]	-	-	RAZ/SBZP.
[30]	TE	RW	Thumb exception Enable: 0 Exceptions including reset are handled in ARM state. 1 Exceptions including reset are handled in Thumb state. The <b>TEINIT</b> signal defines the reset value.
[29:28]	-	-	RAZ/SBZP.
[27]	NMFI	RO	Nonmaskable FIQ support. The bit cannot be configured by software. The <b>CFGNMFI</b> signal defines the reset value.
[26]	-	-	RAZ/SBZP.
[25]	EE	RW	Exception Endianness. This bit determines how the E bit in the CPSR is set on an exception: 0 CPSR E bit is set to 0 on an exception. 1 CPSR E bit is set to 1 on an exception. The <b>CFGEND</b> signal defines the reset value.
[24]	-	-	RAZ/WI.
[23:22]	-	-	RAO/SBOP.
[21]	FI	RW	Fast Interrupts configuration enable bit. This bit can be used to reduce interrupt latency. The permitted values of this bit are: 0 All performance features enabled. This is the reset value. 1 Low interrupt latency configuration. Some performance features disabled.
[20]	-	-	RAZ/SBZP.
[19]	DZ	RW	Divide by Zero fault enable bit. This bit controls whether an integer divide by zero causes an UNDEFINED Instruction exception: 0 Divide by zero returns the result zero, and no exception is taken. This is the reset value. 1 Attempting a divide by zero causes an UNDEFINED Instruction exception on the SDIV or UDIV instruction.
[18]	-	-	RAO/SBOP.
[17]	BR	RW	Background Region bit. When the MPU is enabled this bit controls how an access that does not map to any MPU memory region is handled: 0 Any access to an address that is not mapped to an MPU region generates a Background Fault memory abort. This is the reset value. 1 The default memory map is used as a background region: <ul style="list-style-type: none"> <li>A privileged access to an address that does not map to an MPU region takes the properties defined for that address in the default memory map.</li> <li>An unprivileged access to an address that does not map to an MPU region generates a Background Fault memory abort.</li> </ul>
[16]	-	-	RAO/SBOP.
[15]	-	-	RAZ/SBZP.

Table 4-25 SCTLR bit assignments (continued)

Bits	Name	Access	Function
[14]	-	-	RAZ/WI.
[13]	V	RW	Vectors bit. This bit selects the base address of the exception vectors: 0 Normal exception vectors, base address 0x00000000. 1 High exception vectors, Hivecs, base address 0xFFFF0000. At reset the value of this bit is taken from <b>VINITI</b> .
[12]	I <sup>a</sup>	-	Determines if instructions can be cached at any available cache level: 0 Instruction caching disabled at all levels. This is the reset value. 1 Instruction caching enabled.
[11]	Z	RW	Enables program flow prediction: 0 Program flow prediction disabled. This is the reset value. 1 Program flow prediction enabled.
[10]	SW	RW	SWP/SWPB enable bit: 0 SWP and SWPB are UNDEFINED. This is the reset value. 1 SWP and SWPB perform normally.
[9:7]	-	-	RAZ/SBZP.
[6:3]	-	-	RAO/SBOP.
[2]	C <sup>a</sup>	RW	Determines if data can be cached at any available cache level: 0 Data caching disabled at all levels. This is the reset value. 1 Data caching enabled.
[1]	A	RW	Enables strict alignment of data to detect alignment faults in data accesses: 0 Strict alignment fault checking disabled. This is the reset value. 1 Strict alignment fault checking enabled. Any unaligned access to Device or Strongly Ordered memory generates an alignment fault and therefore does not cause any peripheral interface access. This means that the access examples given in this manual never show unaligned accesses to Device or Strongly Ordered memory.
[0]	M	RW	Enables the MPU: 0 MPU disabled. This is the reset value. 1 MPU enabled.

a. RW if caches implemented, RAZ/WI if no caches.

Attempts to modify read-only bits are ignored.

To access the SCTLR, read or write the CP15 register with:

MRC p15, 0, <Rd>, c1, c0, 0; Read SCTLR

MCR p15, 0, <Rd>, c1, c0, 0; Write SCTLR

#### 4.3.10 Auxiliary Control Register

The ACTLR characteristics are:

##### Purpose

Controls:

- QoS settings.
- ECC checking, if implemented.
- Allocation in one way.

- Automatic data cache coherency.
- Speculative accesses on AXI.
- Broadcast of cache, branch predictor, and maintenance operations.
- Enabling the MRP, if implemented.

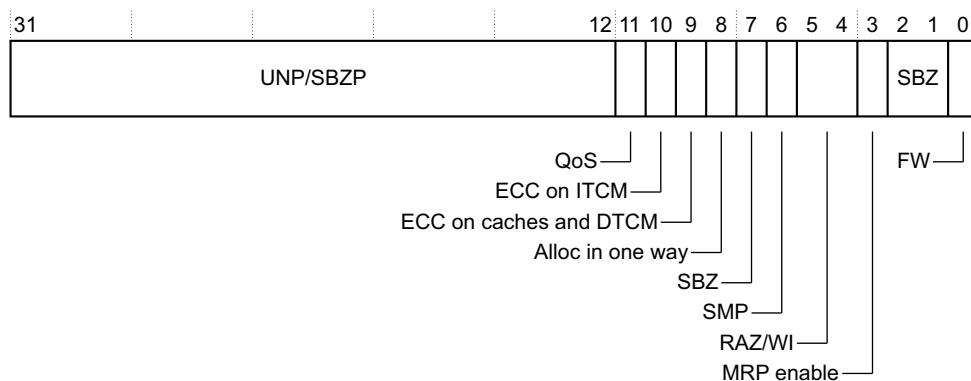
**Usage constraints** The ACTLR is only accessible in privileged mode.

**Configurations** Available in all configurations.

- When the SMP bit = 0, Inner Cacheable Shareable attributes are treated as Non-cacheable.
- When the SMP bit is set:
  - Broadcasting cache maintenance operations is permitted if the FW bit is set.
  - Receiving cache maintenance operations broadcast by other Cortex-R7 processors in the same coherent cluster is permitted if the FW bit is set.
  - The Cortex-R7 processor can send and receive coherent requests for Shareable Inner Write-back Write-Allocate accesses from the other Cortex-R7 processors in the same coherent cluster.

**Attributes** See the register summary in [Table 4-3 on page 4-5](#).

[Figure 4-9](#) shows the ACTLR bit assignments.



**Figure 4-9 ACTLR bit assignments**

[Table 4-26](#) shows the ACTLR bit assignments.

**Table 4-26 ACTLR bit assignments**

Bits	Name	Function
[31:12]	-	UNP/SBZP
[11]	QoS	Quality of Service bit 0 Disabled. This is the reset value. 1 Enabled. See <a href="#">System configurability and QoS on page 8-13</a> .



**Table 4-26 ACTLR bit assignments (continued)**

Bits	Name	Function
[10]	ECC on ITCM	Support for ECC on ITCM, if implemented: 0 Disabled. 1 Enabled. The reset value is defined by the <b>ITCMECCEN</b> signal. If ECC is not implemented this bit is RAZ/WI.
[9]	ECC on caches and DTCM	Support for ECC on instruction and data cache and DTCM, if implemented: 0 Disabled. This is the reset value. 1 Enabled. If ECC is not implemented this bit is RAZ/WI.
[8]	Alloc in one way	Enable allocation in one cache way only. For use with memory copy operations to reduce cache pollution. The reset value is zero.
[7]	-	SBZ
[6]	SMP	Signals if the Cortex-R7 processor is taking part in coherency or not. If this bit is set, then Inner Write Back Shareable is treated as Cacheable. The reset value is zero.
[5:4]	-	RAZ/WI
[3]	MRP enable	MRP enable: 0 Disabled. This is the reset value. 1 Enabled.
[2:1]	-	SBZ
[0]	FW	Cache maintenance broadcast: 0 Disabled. This is the reset value. 1 Enabled. RAZ/WI if only one Cortex-R7 processor is present.

To access the ACTLR, read or write the CP15 register with:

MRC p15, 0, <Rd>, c1, c0, 1; Read ACTLR

MCR p15, 0, <Rd>, c1, c0, 1; Write ACTLR

#### 4.3.11 Coprocessor Access Control Register

The CPACR characteristics are:

- Purpose**
- Sets access rights for the coprocessors CP11 and CP10.
  - Enables software to determine if any particular coprocessor exists in the system.

**Note**

This register has no effect on access to CP14 or CP15.

**Usage constraints** The CPACR is only accessible in privileged mode.

**Configurations** Available in all configurations.

**Attributes** See the register summary in [Table 4-3 on page 4-5](#).

[Figure 4-10 on page 4-28](#) shows the CPACR bit assignments.

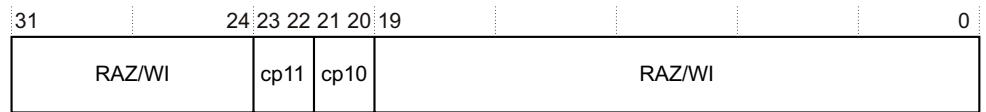


Figure 4-10 CPACR bit assignments

Table 4-27 shows the CPACR bit assignments.

Table 4-27 CPACR bit assignments

Bits	Name	Function
[31:24]	-	RAZ/WI.
[23:22]	cp11	Defines access permissions for CP11: <div> <div>0b00</div> <div>Access denied. This is the reset value, and the behavior for nonexistent coprocessors. Attempted access generates an Undefined Instruction exception.</div> </div> <div> <div>0b01</div> <div>Privileged mode access only.</div> </div> <div> <div>0b10</div> <div>Reserved.</div> </div> <div> <div>0b11</div> <div>Privileged and user mode access.</div> </div>
[21:20]	cp10	Defines access permissions for CP10: <div> <div>0b00</div> <div>Access denied. This is the reset value, and the behavior for nonexistent coprocessors. Attempted access generates an Undefined Instruction exception.</div> </div> <div> <div>0b01</div> <div>Privileged mode access only.</div> </div> <div> <div>0b10</div> <div>Reserved.</div> </div> <div> <div>0b11</div> <div>Privileged and user mode access.</div> </div>
[19:0]	-	RAZ/WI.

To access the CPACR, read or write the CP15 register with:

```
MRC p15, 0, <Rd>, c1, c0, 2; Read Coprocessor Access Control Register
MCR p15, 0, <Rd>, c1, c0, 2; Write Coprocessor Access Control Register
```

You must execute an ISB immediately after an update of the CPACR. See *Memory Barriers* in the *ARM® Architecture Reference Manual ARMv7-A and ARMv7-R edition*. You must not attempt to execute any instructions that are affected by the change of access rights between the ISB and the register update.

To determine if any particular coprocessor exists in the system, write the access bits for the coprocessor of interest with 0b11. If the coprocessor does not exist in the system the access rights remain set to 0b00.

#### ———— Note ————

You must enable both CP10 and CP11 before accessing any VFP system registers. If the access control bits are programmed differently for CP10 and CP11, operation of VFP features is UNPREDICTABLE. This behavior is applicable for both FPU modes, that is, full or optimized.

### 4.3.12 MPU memory region programming registers

The MPU memory region programming registers program the MPU regions.

There is one register that specifies which set of region registers is to be accessed. See *MPU Memory Region Number Registers* on page 4-32. Each region has its own register to specify:

- Region base address.

- Region size and enable.
- Region access control.

You can implement the processor with 12 or 16 regions.

---

**Note**


---

- When the MPU is enabled:
  - The MPU determines the access permissions for all accesses to memory, including the TCMs. Therefore, you must ensure that the memory regions in the MPU are programmed to cover the complete TCM address space with the appropriate access permissions. You must define at least one of the regions in the MPU.
  - An access to an UNDEFINED area of memory generates a background fault.
- For the TCM space, the processor uses the access permissions but ignores the region attributes from MPU.

CP15, c9 sets the location of the TCM base address. For more information see [DTCM Region Register on page 4-33](#) and [ITCM Region Register on page 4-34](#).

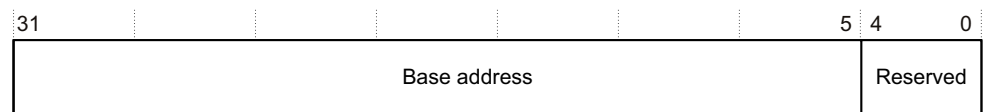
---

## MPU Region Base Address Registers

The DRBAR characteristics are:

<b>Purpose</b>	Describe the base address of the region specified by the RGNR. The region base address must always align to the region size.
<b>Usage constraints</b>	The DRBAR are only accessible in privileged mode.
<b>Configurations</b>	Available in all configurations.
<b>Attributes</b>	See the register summary in <a href="#">Table 4-5 on page 4-6</a> .

[Figure 4-11](#) shows the DRBAR bit assignments.



**Figure 4-11 DRBAR bit assignments**

[Table 4-28](#) shows the DRBAR bit assignments.

**Table 4-28 DRBAR bit assignments**

Bits	Name	Function
[31:5]	Base address	Physical base address. Defines the base address of a region.
[4:0]	Reserved	SBZ

To access the DRBAR, read or write the CP15 register with:

MRC p15, 0, <Rd>, c6, c1, 0 ; Read MPU Region Base Address Register  
 MCR p15, 0, <Rd>, c6, c1, 0 ; Write MPU Region Base Address Register

## MPU Region Size and Enable Registers

The DRSR characteristics are:

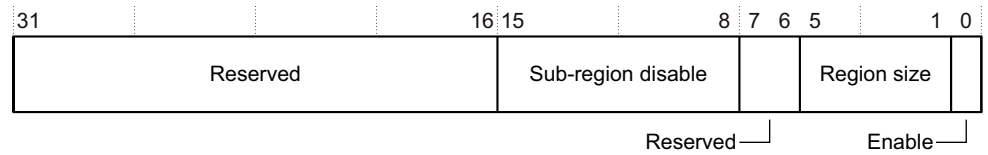
- Purpose**
- Specify the size of the region specified by the RGNR.
  - Identify the address ranges that are used for a particular region.
  - Enable or disable the region, and its sub-regions, specified by the RGNR.

**Usage constraints** The DRSR are only accessible in privileged mode.

**Configurations** Available in all configurations.

**Attributes** See the register summary in [Table 4-5 on page 4-6](#).

[Figure 4-12](#) shows the DRSR bit assignments.



**Figure 4-12 DRSR bit assignments**

[Table 4-29](#) shows the DRSR bit assignments.

**Table 4-29 DRSR bit assignments**

Bits	Name	Function																																																				
[31:16]	Reserved	SBZ.																																																				
[15:8]	Sub-region disable	Each bit position represents a sub-region, 0-7 <sup>a</sup> . Bit[8] corresponds to sub-region 0 ... Bit[15] corresponds to sub-region 7. The meaning of each bit is: 0                   Address range is part of this region. 1                   Address range is not part of this region.																																																				
[7:6]	Reserved	SBZ.																																																				
[5:1]	Region size	Defines the region size: <table><tr><td>0b01110</td><td>32KB.</td><td>0b10111</td><td>16MB.</td></tr><tr><td>0b00000-0b00110</td><td>UNPREDICTABLE.</td><td>0b01111</td><td>64KB.</td><td>0b11000</td><td>32MB.</td></tr><tr><td>0b00111</td><td>256 bytes.</td><td>0b10000</td><td>128KB.</td><td>0b11001</td><td>64MB.</td></tr><tr><td>0b01000</td><td>512 bytes.</td><td>0b10001</td><td>256KB.</td><td>0b11010</td><td>128MB.</td></tr><tr><td>0b01001</td><td>1KB.</td><td>0b10010</td><td>512KB.</td><td>0b11011</td><td>256MB.</td></tr><tr><td>0b01010</td><td>2KB.</td><td>0b10011</td><td>1MB.</td><td>0b11100</td><td>512MB.</td></tr><tr><td>0b01011</td><td>4KB.</td><td>0b10100</td><td>2MB.</td><td>0b11101</td><td>1GB.</td></tr><tr><td>0b01100</td><td>8KB.</td><td>0b10101</td><td>4MB.</td><td>0b11110</td><td>2GB.</td></tr><tr><td>0b01101</td><td>16KB.</td><td>0b10110</td><td>8MB.</td><td>0b11111</td><td>4GB.</td></tr></table>	0b01110	32KB.	0b10111	16MB.	0b00000-0b00110	UNPREDICTABLE.	0b01111	64KB.	0b11000	32MB.	0b00111	256 bytes.	0b10000	128KB.	0b11001	64MB.	0b01000	512 bytes.	0b10001	256KB.	0b11010	128MB.	0b01001	1KB.	0b10010	512KB.	0b11011	256MB.	0b01010	2KB.	0b10011	1MB.	0b11100	512MB.	0b01011	4KB.	0b10100	2MB.	0b11101	1GB.	0b01100	8KB.	0b10101	4MB.	0b11110	2GB.	0b01101	16KB.	0b10110	8MB.	0b11111	4GB.
0b01110	32KB.	0b10111	16MB.																																																			
0b00000-0b00110	UNPREDICTABLE.	0b01111	64KB.	0b11000	32MB.																																																	
0b00111	256 bytes.	0b10000	128KB.	0b11001	64MB.																																																	
0b01000	512 bytes.	0b10001	256KB.	0b11010	128MB.																																																	
0b01001	1KB.	0b10010	512KB.	0b11011	256MB.																																																	
0b01010	2KB.	0b10011	1MB.	0b11100	512MB.																																																	
0b01011	4KB.	0b10100	2MB.	0b11101	1GB.																																																	
0b01100	8KB.	0b10101	4MB.	0b11110	2GB.																																																	
0b01101	16KB.	0b10110	8MB.	0b11111	4GB.																																																	
[0]	Enable	Enables or disables a memory region: 0                   Memory region disabled. Memory regions are disabled on reset. 1                   Memory region enabled. A memory region must be enabled before it is used.																																																				

a. Sub-region 0 covers the least significant addresses in the region, while sub-region 7 covers the most significant addresses in the region. For more information, see [Subregions on page 8-4](#).

To access the DRSR, read or write the CP15 register with:

MRC p15, 0, <Rd>, c6, c1, 2 ; Read Data MPU Region Size and Enable Register  
MCR p15, 0, <Rd>, c6, c1, 2 ; Write Data MPU Region Size and Enable Register

Writing a region size that is outside the range results in UNPREDICTABLE behavior.

## MPU Region Access Control Registers

The DRACR characteristics are:

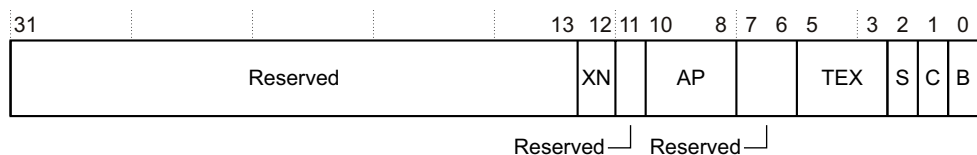
**Purpose** Hold the region attributes and access permissions for the region specified by the RGNR.

**Usage constraints** The DRACR are only accessible in privileged mode.

**Configurations** Available in all configurations.

**Attributes** See the register summary in [Table 4-5 on page 4-6](#).

[Figure 4-13](#) shows the DRACR bit assignments.



**Figure 4-13 DRACR bit assignments**

[Table 4-30](#) shows the DRACR bit assignments.

**Table 4-30 DRACR bit assignments**

Bits	Name	Function
[31:13]	Reserved	SBZ.
[12]	XN	Execute never. Determines if a region of memory is executable: 0 All instruction fetches enabled. 1 No instruction fetches enabled.
[11]	Reserved	SBZ
[10:8]	AP	Access permission. Defines the data access permissions. For more information on AP bit values, see <a href="#">Table 4-31 on page 4-32</a> .
[7:6]	Reserved	SBZ.
[5:3]	TEX	Type extension. Defines the type extension attribute <sup>a</sup> .
[2]	S	Share. Determines if the memory region is Shareable or Non-shareable: 0 Non-shareable. 1 Shareable. This bit only applies to Normal, not Device or Strongly Ordered memory.
[1]	C	C bit <sup>a</sup> :
[0]	B	B bit <sup>a</sup> :

a. For more information on this region attribute, see [Table 8-2 on page 8-8](#).

Table 4-31 shows the AP bit values that determine the permissions for privileged and user data access.

### Table 4-31 Access data permission bit encoding

AP bit values	Privileged permissions	User permissions	Description
0b000	No access	No access	All accesses generate a permission fault
0b001	Read/write	No access	Privileged access only
0b010	Read/write	Read-only	Writes in user mode generate permission faults
0b011	Read/write	Read/write	Full access
0b100	UNP	UNP	Reserved
0b101	Read-only	No access	Privileged read-only
0b110	Read-only	Read-only	Privileged/user read-only
0b111	UNP	UNP	Reserved

To access the DRACR, read or write the CP15 register with:

```
MRC p15, 0, <Rd>, c6, c1, 4 ; Read Region Access Control Register
MCR p15, 0, <Rd>, c6, c1, 4 ; Write Region Access Control Register
```

To execute instructions in user and privileged modes:

- The region must have read access as defined by the AP bits.
- The XN bit must be set to 0.

## MPU Memory Region Number Registers

The RGNR characteristics are:

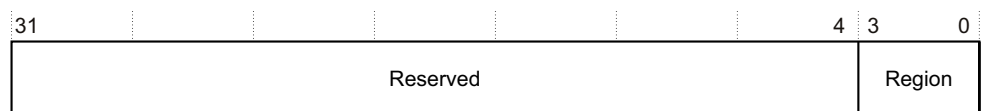
<b>Purpose</b>	Determines which register is accessed. There is one register for each implemented memory region.
----------------	--

**Usage constraints** The RGNR are only accessible in privileged mode.

<b>Configurations</b>	Available in all configurations.
-----------------------	----------------------------------

**Attributes** See the register summary in [Table 4-5 on page 4-6](#).

Figure 4-14 shows the RGNR bit assignments.



### Figure 4-14 RGNR bit assignments

Table 4-32 shows the RGNR bit assignments.

**Table 4-32 RGNR bit assignments**

Bits	Name	Function
[31:4]	Reserved	SBZ.
[3:0]	Region	Defines the group of registers to be accessed. Read the MPU Type Register to determine the number of supported regions, see <a href="#">MPU Type Register on page 4-17</a> .

To access the RGNR, read or write the CP15 register with:

MRC p15, 0, <Rd>, c6, c2, 0 ; Read MPU Memory Region Number Register  
MCR p15, 0, <Rd>, c6, c2, 0 ; Write MPU Memory Region Number Register

Writing this register with a value greater than or equal to the number of regions from the MPU Type Register is UNPREDICTABLE. Associated register bank accesses are also UNPREDICTABLE.

### 4.3.13 DTCM Region Register

The DTCMRR characteristics are:

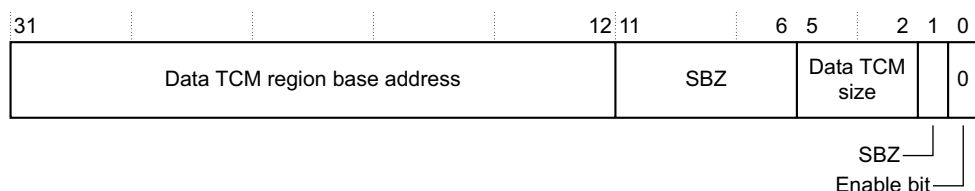
**Purpose** Indicates the base address and size of the Data TCM.

**Usage constraints** The DTCMRR is only accessible in privileged mode.

**Configurations** Available in all configurations.

**Attributes** See the register summary in [Table 4-7 on page 4-8](#).

[Figure 4-15](#) shows the DTCMRR bit assignments.



**Figure 4-15 DTCMRR bit assignments**

Table 4-33 shows the DTCMRR bit assignments.

**Table 4-33 DTCMRR bit assignments**

Bits	Name	Function
[31:12]	Data TCM region base address	Indicates the Data TCM region base address. The reset value is 0.
[11:6]	-	SBZ
[5:2]	Data TCM size	Indicates the Data TCM region size: 0b0000      0KB. 0b0011      4KB. 0b0100      8KB. 0b0101      16KB. 0b0110      32KB. 0b0111      64KB. 0b1000      128KB. All other values are Reserved.
[1]	-	SBZ
[0]	Enable bit	Enable bit: 0            Disabled. This is the reset value. 1            Enabled. If Data TCM is not implemented, this field is Read-only and its value is 0.

To access the DTCMRR, read or write the CP15 register with:

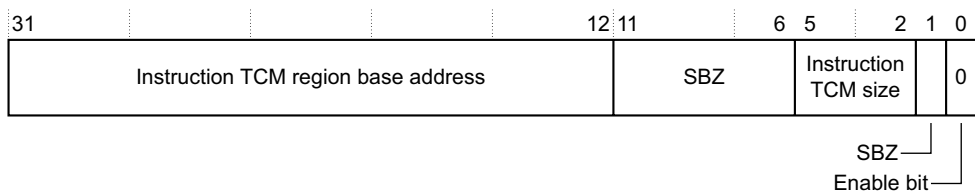
MRC p15, 0, <Rd>, c9, c1, 0; Read DTCM Region Register  
 MCR p15, 0, <Rd>, c9, c1, 0; Write DTCM Region Register

#### 4.3.14 ITCM Region Register

The ITCMRR characteristics are:

- Purpose**
- Indicates the base address and size of the Instruction TCM.
  - Enables the Instruction TCM directly from reset.
- Usage constraints** The ITCMRR is :
- Only accessible in privileged mode.
  - For use with **INITRAM0**.
- Configurations** Available in all configurations.
- Attributes** See the register summary in [Table 4-7 on page 4-8](#).

Figure 4-16 shows the ITCMRR bit assignments.



**Figure 4-16 ITCMRR bit assignments**



Table 4-34 shows the ITCMRR bit assignments.

**Table 4-34 ITCMRR bit assignments**

Bits	Name	Function
[31:12]	Instruction TCM region base address	Indicates the Instruction TCM region base address. When <b>INITRAM0</b> is HIGH and <b>VINITH0</b> is HIGH for processor 0, the reset value is 0xFFFF0, otherwise the reset value is 0x00000. The same applies for processor 1, if present.
[11:6]	-	SBZ
[5:2]	Instruction TCM size	Indicates the Instruction TCM region size: 0b0000      0KB. 0b0011      4KB. 0b0100      8KB. 0b0101      16KB. 0b0110      32KB. 0b0111      64KB. 0b1000      128KB. All other values are Reserved.
[1]	-	SBZ
[0]	Enable bit	Enable bit: 0              Disabled. This is the reset value. 1              Enabled. When <b>INITRAM0</b> is HIGH this enables the Instruction TCM for processor 0 directly from reset. The same applies for processor 1, if present. When <b>INITRAM1</b> is HIGH, this enables the Instruction TCM for processor 1 directly from reset. If Instruction TCM is not implemented, this field is Read-only and its value is 0.

To access the ITCMRR, read or write the CP15 register with:

MRC p15, 0, <Rd>, c9, c1, 1; Read ITCM Region Register  
MCR p15, 0, <Rd>, c9, c1, 1; Write ITCM Region Register

#### 4.3.15 Power Control Register

The PCR characteristics are:

<b>Purpose</b>	Enables you to set dynamic clock gating.
<b>Usage constraints</b>	There are no usage constraints.
<b>Configurations</b>	Available in all configurations.
<b>Attributes</b>	See the register summary in <a href="#">Table 4-9 on page 4-9</a> .

[Figure 4-17 on page 4-36](#) shows the PCR bit assignments.

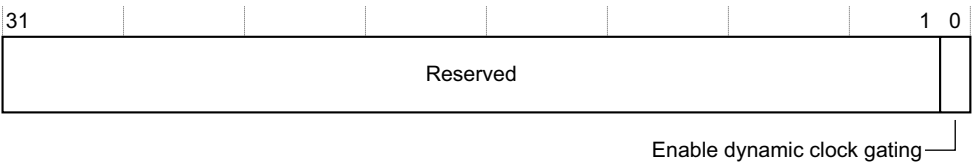


Figure 4-17 PCR bit assignments

Table 4-35 shows the PCR bit assignments.

Table 4-35 PCR bit assignments

Bits	Name	Function
[31:1]	-	Reserved.
[0]	Enable dynamic clock gating	Disabled at reset.

To access the Power Control Register, read or write the CP15 register with:

```
MRC p15, 0, <Rd>, c15, c0, 0; Read Power Control Register
MCR p15, 0, <Rd>, c15, c0, 0; Write Power Control Register
```

4.3.16 Cache and TCM Debug Operation Register

The CTDOR characteristics are:

- Purpose** Describes the access operation required for cache and TCM debug.
- Usage constraints** The CTDOR is write accessible in privileged mode only.
- Configurations** Available in all configurations.
- Attributes** See the register summary in Table 4-9 on page 4-9.

Figure 4-18 shows CTDOR bit assignments for cache RAMs.

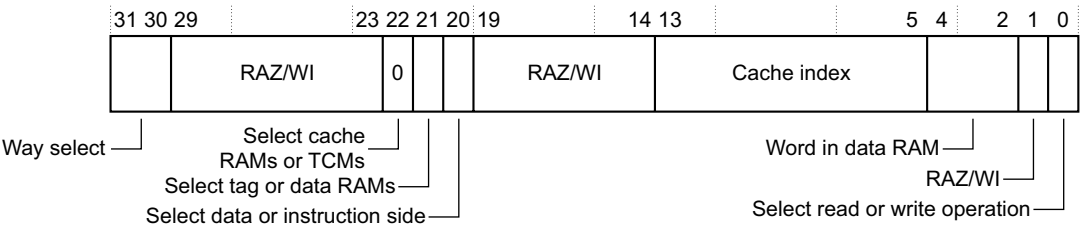


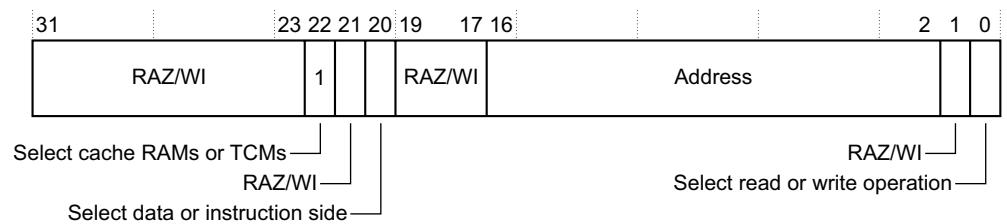
Figure 4-18 CTDOR bit assignments for cache RAMs

Table 4-36 shows the CTDOR bit assignments for cache RAMs.

**Table 4-36 CTDOR bit assignments for cache RAMs**

Bits	Name	Function	
[31:30]	Way selector	Indicates the way to select in the cache.	
		0b00	Way 0.
		0b01	Way 1.
		0b10	Way 2.
		0b11	Way 3.
[29:23]	-	RAZ/WI.	
[22]	Select cache RAMs or TCMs	0	Use with cache RAMs.
[21]	Select tag or data RAMs	0	Use with tag RAMs.
		1	Use with data RAMs.
[20]	Select data or instruction side	0	Select data side.
		1	Select instruction side.
[19:14]	-	RAZ/WI.	
[13:5]	Cache index	Indicates the cache index.	
[4:2]	Word in data RAM	Indicates the 32-bit word in the cache line. Only required if accessing data RAM.	
[1]	-	RAZ/WI.	
[0]	Select read or write operation	0	Read operation.
		1	Write operation.

Figure 4-19 shows the CTDOR bit assignments for TCM RAMs.



**Figure 4-19 CTDOR bit assignments for TCMs**

Table 4-37 shows the CTDOR bit assignments for TCM RAMs.

**Table 4-37 CTDOR bit assignments for TCMs**

Bits	Name	Function	
[31:23]	-	RAZ/WI.	
[22]	Select cache or TCM RAMs	1	Use with TCMs.
[21]	-	RAZ/WI.	
[20]	Select data or instruction side	0	Select data side.
		1	Select instruction side.

Table 4-37 CTDOR bit assignments for TCMs (continued)

Bits	Name	Function
[19:17]	-	RAZ/WI.
[16:2]	Address	Indicates the address of the TCM RAM.
[1]	-	RAZ/WI.
[0]	Select read or write operation	0 Read operation.
		1 Write operation.

To access the CTDOR, read or write the CP15 register with:

MRC p15, 0, <Rd>, c15, c1, 0 ; Read Cache and TCM Debug Operation Register

MCR p15, 0, <Rd>, c15, c1, 0 ; Write Cache and TCM Debug Operation Register

### 4.3.17 RAM Access Data Registers

The RADRLO and RADRHI characteristics are:

**Purpose**            **Read**      Returns the 32-bit data value for the debug operation.  
**Write**            Sets the value to be written by the direct RAM access operation.

———— **Note** —————

Accesses to data cache use only RADRLO. RADRHI is RAZ/WI.

Accesses to instruction cache use both RADRLO and RADRHI because these accesses require 64-bit values.

**Usage constraints**    The RADRLO and RADRHI are only accessible in privileged mode.

**Configurations**      Available in all configurations.

**Attributes**            See the register summary in [Table 4-9 on page 4-9](#).

[Figure 4-20](#) shows the RADRLO bit assignments.

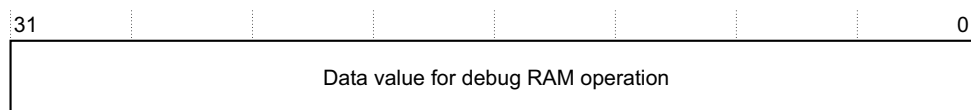


Figure 4-20 RADRLO bit assignments

[Figure 4-21](#) shows the RADRHI bit assignments.

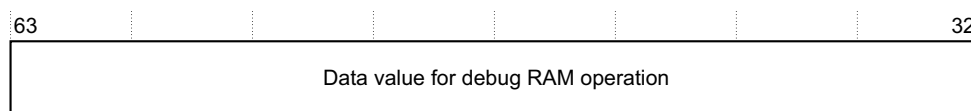


Figure 4-21 RADRHI bit assignments

To access the RADRLO, read or write the CP15 register with:

MRC p15, 0, <Rd>, c15, c1, 1; Read the CP15 debug cache/tcm access data register

MCR p15, 0, <Rd>, c15, c1, 1; Write the CP15 debug cache/tcm access data register

To access the RADRHI, read or write the CP15 register with:

MRC p15, 0, <Rd>, c15, c1, 2; Read the CP15 debug cache/tcm access data register  
MCR p15, 0, <Rd>, c15, c1, 2; Write the CP15 debug cache/tcm access data register

### 4.3.18 RAM Access ECC Register

The RAECCR characteristics are:

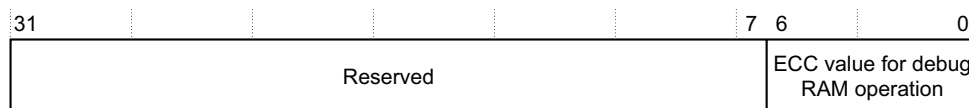
**Purpose**                      **Read**      Returns the ECC chunk value selected according to the operation register, and associated with the selected data value.  
**Write**                      Sets the ECC chunk to be written by the next direct ram access operation.

**Usage constraints**      The RAECCR is only accessible in privileged mode.

**Configurations**          Available only if ECC is implemented.

**Attributes**                See the register summary in [Table 4-9 on page 4-9](#)

[Figure 4-22](#) shows the RAECCR bit assignments.



**Figure 4-22 RAECCR bit assignments**

[Table 4-38](#) shows the RAECCR bit assignments.

**Table 4-38 RAECCR bit assignments**

Bits	Name	Function
[31:7]	-	RAZ/WI
[6:0]	ECC value for debug RAM operation	ECC chunk value for the selected debug operation.

To access the RAECCR, read or write the CP15 register with:

MRC p15, 0, <Rd>, c15, c1, 3 ; Read the ECC chunk value for the debug operation  
MCR p15, 0, <Rd>, c15, c1, 3 ; Write the ECC chunk value for the debug operation

### 4.3.19 ECC Error Registers

There are four banks of ECC Error Registers:

- One bank for the data cache.
- One bank for the instruction cache.
- One bank for the data TCM.
- One bank for the instruction TCM.

Banks for data cache and instruction caches have three entries, DEER0-2/IEER0-2. Banks for data TCM and instruction TCM have one entry, DTCMEER/ITCMEER.

The DEER0-2/IEER0-2 and DTCMEER/ITCMEER characteristics are:

**Purpose**                      Indicate where ECC errors have occurred.

**Usage constraints**      The DEER0-2/IEER0-2 and DTCMEER/ITCMEER are only accessible in privileged mode.

**Configurations** Available in all configurations.

**Attributes** See the register summary in [Table 4-9 on page 4-9](#).

[Table 4-39](#) shows the DEER0-2 bit assignments.

**Table 4-39 DEER0-2 bit assignments**

Bits	Function
[31:28]	Way affected in one-hot encoding.
[27:26]	Reserved. RAZ/WI.
[25]	A fatal error has occurred.
[24:19]	Reserved. RAZ/WI.
[18]	Error occurred in Data RAM.
[17]	Error occurred in Tag RAM.
[16]	Error occurred in SCU RAM.
[15:14]	Reserved. RAZ/WI.
[13:5]	Faulty address where error has occurred.
[4:2]	Faulty word affected by ECC error.
[1]	Error is hard, that is, only software can write on it.
[0]	This entry contains valid information.

[Table 4-40](#) shows the IEER0-2 bit assignments.

**Table 4-40 IEER0-2 bit assignments**

Bits	Function
[31:28]	Way affected in one-hot encoding.
[27:26]	Reserved. RAZ/WI.
[25]	A fatal error has occurred.
[24:14]	Reserved. RAZ/WI.
[13:5]	Faulty address where error has occurred.
[4:2]	Faulty word affected by ECC error.
[1]	Error is hard, that is, only software can write on it.
[0]	This entry contains valid information.

Table 4-41 shows the DTCMEER/ITCMEER bit assignments.

**Table 4-41 DTCMEER/ITCMEER bit assignments**

Bits	Function
[31:26]	Reserved. RAZ/WI.
[25]	A fatal error has occurred.
[24:17]	Reserved. RAZ/WI.
[16:2]	Faulty address where error has occurred.
[1]	Error is hard, that is, only software can write on it.
[0]	This entry contains valid information.

When an error is detected by the processor circuitry, the first available ECC Error Register is updated with the memory information of where the error has been found, for example, index, way, or memory type, and bit [0] is set to 1. Bit [1] can only be accessed by the software and is used to mark entries where there are hard errors. Writing zeros to bits [25] and [1:0] enables their contents to be reset. Resetting the ECC information is therefore possible by writing these six registers successively, two for each entry.

If bit [16] is set to 1, the error has been detected by the SCU, therefore this index-way pair is also present in the SCU error bank. When resetting an entry, if this bit is set, you must clear the corresponding SCU entry to maintain the coherence between the SCU error bank and the D-side error bank. This bit is only accessible for the Data Side.

Table 4-42 shows the meaning of the error status bits.

**Table 4-42 Error status bit encoding**

Bits[1:0]	Meaning
00	No error. Entry available, no index masking, not yet processed by the external system.
01	Unanalyzed error. Entry not available, index masking, not yet processed by the external system. It can be either soft or hard. This state is entered when any ECC error occurs.
10	Not used.
11	Permanent error. Entry not available, index masking, already processed by the external system.

All errors captured between two analyses of the global monitor or left in the bank by it are visible in this bank, as long as the bank has not been filled.

To access the DEER0-2, read or write the CP15 register with:

MRC p15, 0, <Rd>, c15, c2, n; Read ECC entry no. n, n in set [0, 1, 2]  
MCR p15, 0, <Rd>, c15, c2, n; Write ECC entry no. n, n in set [0, 1, 2]

To access the IEER0-2, read or write the CP15 register with:

MRC p15, 0, <Rd>, c15, c3, n; Read ECC entry no. n, n in set [0, 1, 2]  
MCR p15, 0, <Rd>, c15, c3, n; Write ECC entry no. n, n in set [0, 1, 2]

To access the DTCMEER, read or write the CP15 register with:

MRC p15, 0, <Rd>, c15, c4, 0; Read TCM ECC entry  
MCR p15, 0, <Rd>, c15, c4, 0; Write TCM ECC entry

To access the ITCMEER, read or write the CP15 register with:

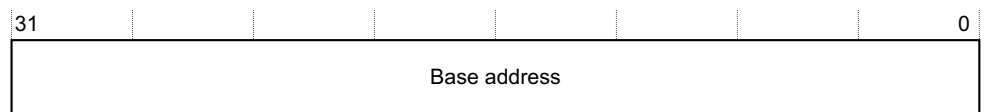
MRC p15, 0, <Rd>, c15, c5, 0; Read TCM ECC entry  
MCR p15, 0, <Rd>, c15, c5, 0; Write TCM ECC entry

#### 4.3.20 Configuration Base Address Register

The CBAR characteristics are:

<b>Purpose</b>	Holds the physical base address of the memory-mapped interrupt controller registers.
<b>Usage constraints</b>	The CBAR is a read-only register.
<b>Configurations</b>	Reset to <b>PERIPHBASE[31:13]</b> so that software can determine the location of the private memory region.
<b>Attributes</b>	See the register summary in <a href="#">Table 4-9 on page 4-9</a> .

[Figure 4-23](#) shows the CBAR bit assignments.



**Figure 4-23 CBAR bit assignments**

To access the CBAR, read the CP15 register with:

MRC p15, 4, <Rd>, c15, c0, 0; Read Configuration Base Address Register

#### 4.3.21 Using the CTDOR

[Example 4-1](#) shows how to use the *Cache and TCM Debug Operation Register* (CTDOR).

##### Example 4-1 Corrupting a single bit in word x of a data cache

Assumptions:

- The data to be corrupted is present in L1, way 1. Ways are numbered 0-3.
- Word 2. Words are numbered 0-7.
- The address of the data is stored in r0.

Use the register as follows:

1. Write the CTDOR:

```
MOV r2,    #0x3F00    ; Index mask (high bits)
ORR r2, r2, #0xE0     ; Index mask (low bits)
AND r1, r0, r2        ; Index extraction for cache access
ORR r1, r1, #1<<30    ; Way indication
ORR r1, r1, #1<<22    ; Cache RAM selection
ORR r1, r1, #1<<21    ; Data RAM selection
ORR r1, r1, #0<<20    ; Data side memory selection
ORR r1, r1, #2<<2     ; Word selection
ORR r1, r1, #0<<0     ; Read operation
```

MCR p15, 0, r1, c15, c1, 0 ; Write CTDOR with operation selected above

2. The word to corrupt is now stored in c15,0,c1,1 and its ECC chunk in c15,0,c1,2:



```

MRC p15, 0, r3, c15, c1, 1 ; Read data
MRC p15, 0, r4, c15, c1, 3 ; Read ECC chunk (useless in this case)
ORR r3, r3, #1<<5          ; We want to corrupt bit 5 in read word
MRC p15, 0, r3, c15, c1, 1 ; Copy corrupted data to CP15 register
MRC p15, 0, r4, c15, c1, 3 ; Copy corrupted ECC chunk to CP15 register (useless
in this case)

```

3. Write the CTDOR to write the corrupted word on the data cache:

```

ORR r1, r1, #1<<0          ; Write operation
MCR p15, 0, r1, c15, c1, 0 ; Actual write of corrupted data and chunk to L1 cache

```

---

# Chapter 5

## Floating Point Unit Programmers Model

This chapter describes the programmers model of the optional *Floating-Point Unit* (FPU). It contains the following sections:

- [\*About the FPU programmers model\*](#) on page 5-2.
- [\*IEEE 754 standard compliance\*](#) on page 5-3.
- [\*Instruction throughput and latency\*](#) on page 5-4.
- [\*Register summary\*](#) on page 5-6.
- [\*Register descriptions\*](#) on page 5-8.

## 5.1 About the FPU programmers model

The FPU implements the VFPv3-D16 architecture and the Common VFP Sub-Architecture v2. This includes the instruction set of the VFPv3 architecture. See the *ARM® Architecture Reference Manual ARMv7-A and ARMv7-R edition* for information on the VFPv3 instruction set.

### 5.1.1 FPU functionality

The FPU is an implementation of the *ARM Vector Floating Point v3* architecture with 16 double-precision registers or 32 single-precision registers, VFPv3-D16. It provides floating-point computation functionality that is compliant with the *ANSI/IEEE Std 754-1985, IEEE Standard for Binary Floating-Point Arithmetic*, referred to as the IEEE 754 standard. The FPU supports all data-processing instructions and data types in the VFPv3 architecture as described in the *ARM® Architecture Reference Manual ARMv7-A and ARMv7-R edition*.

The FPU fully supports add, subtract, multiply, divide, multiply and accumulate, and square root operations. It also provides conversions between fixed-point and floating-point data formats, and floating-point constant instructions. The FPU does not support any data processing operations on vectors in hardware. Any data processing instruction that operates on a vector generates an UNDEFINED exception. The operation can then be emulated in software if necessary.

## 5.2 IEEE 754 standard compliance

This section introduces issues related to the IEEE 754 standard compliance:

- Hardware and software components.
- Software-based components and their availability.

### 5.2.1 Implementation of the IEEE 754 standard

The following operations from the IEEE 754 standard are not supported by the FPU instruction set:

- Remainder.
- Round floating-point number to integer-valued floating-point number
- Binary-to-decimal conversions.
- Decimal-to-binary conversions.
- Direct comparison of single-precision and double-precision values.

### 5.2.2 IEEE 754 standard implementation choices

Some of the implementation choices permitted by the IEEE 754 standard and used in the VFPv3 architecture are described in the *ARM® Architecture Reference Manual ARMv7-A and ARMv7-R edition*.

### 5.2.3 Supported formats

The Cortex-R7 MPCore processor supports two build options for the FPU:

- An optimized FPU is single-precision and half-precision.
- A full FPU is single-precision, half-precision, and double-precision.

## 5.3 Instruction throughput and latency

Complex instruction dependencies and memory system interactions make it impossible to describe the exact cycle timing for all instructions in all circumstances. The timing described in [Table 5-1](#) is accurate in most cases. For precise timing, you must use a cycle-accurate model of your processor.

### 5.3.1 Definitions of throughput and latency

The definitions of throughput and latency are:

**Throughput** Throughput is the number of cycles after issue that another instruction can begin execution.

**Latency** Latency is the number of cycles after which the data is available for another operation. The forward latency, Fwd, is relevant for *Read-After-Write* (RAW) hazards. The writeback latency, Wbck, is relevant for *Write-After-Write* (WAW) hazards. See [Table 5-1](#).

Latency values assume that the instruction has been issued and that neither the FPU pipeline nor the Cortex-R7 MPCore processor pipeline is stalled.

[Table 5-1](#) shows:

- The FPU instruction throughput and latency cycles for all operations except loads, stores, and system register accesses.
- The old ARM assembler mnemonics and the ARM *Unified Assembler Language* (UAL) mnemonics.

**Table 5-1 FPU instruction throughput and latency cycles**

Old ARM assembler mnemonic	UAL	Single-precision		Double-precision	
		Throughput	Latency		Throughput
			Fwd	Wbck	
FADD	VADD	1	4		1
FSUB	VSUB				4
FCVT	VCVT				
FSHTOD, FSHTOS					
FSITOD, FSITOS					
FTOSHD, FTOSHS					
FTOSID, FTOSIS					
FTOSL, FTOUH					
FTOUI{Z}D, FTOUI{Z}S					
FTOULD, FTOULS, FUHTOD, FUHTOS					
FUITOD, FUITOS					
FULTOD, FULTOS					
FMUL	VMUL	1	4	2	6
FNMUL	VNMUL				

Table 5-1 FPU instruction throughput and latency cycles (continued)

Old ARM assembler mnemonic	UAL	Single-precision			Double-precision		
		Throughput	Latency		Throughput	Latency	
			Fwd	Wbck		Fwd	Wbck
FMAC	VMLA	1	7		2	9	
FNMAC	VMLS						
FMSC	VNMLS						
FNMSC	VNMLA						
FCPY	VMOV	1	1	2	1	1	2
FABS	VABS						
FNEG	VNEG						
FCONST	VMOV						
FMRS <sup>a</sup>	VMOV	1	-	3	1	-	3
FMRR(S/D)		2					
FMRD(L/H)		1					
FMSR <sup>b</sup>	VMOV	1	1	2	1	1	2
FM(S/D)RR							
FMD(L/H)R							
FMSTAT	VMRS	1	-	3	1	-	3
FDIV	VDIV	10	15		20	25	
FSQRT	VSQRT	13	17		28	32	
FCMP	VCMP	1	1	4	1	1	4
FCMPE	VCMP{E}						
FCMPZ	VCMP{E}						
FCMPEZ	VCMP{E}						
-	FCVT(T/B) .F16.F32	1	2	2	-	-	-
-	FCVT(T/B) .F32.F16	1	-	4	-	-	-

a. FPU to ARM.

b. ARM to FPU.

## 5.4 Register summary

Table 5-2 shows the FPU system registers. All FPU system registers are 32-bit wide. Reserved register addresses are RAZ/WI.

**Table 5-2 FPU system registers**

Name	Type	Reset	Description
FPSID	RO	0x41023170	See <i>Floating-Point System ID Register</i> on page 5-8
FPSCR	RW	0x00000000	See <i>Floating-Point Status and Control Register</i> on page 5-8
MVFR1	RO	0x01000011	See the <i>ARM® Architecture Reference Manual ARMv7-A and ARMv7-R edition</i>
MVFR0	RO	0x10110221 <sup>a</sup> 0x10110021 <sup>b</sup>	See the <i>ARM® Architecture Reference Manual ARMv7-A and ARMv7-R edition</i>
FPEXC	RW	0x00000000	See <i>Floating-Point Exception Register</i> on page 5-10

a. For full FPU implementation, with double precision.

b. For single-precision FPU implementation.

### 5.4.1 Processor modes for accessing the FPU system registers

Table 5-3 shows the processor modes for accessing the FPU system registers.

**Table 5-3 Accessing FPU system registers**

Register	Privileged access		User access	
	FPEXC EN=0	FPEXC EN=1	FPEXC EN=0	FPEXC EN=1
FPSID	Permitted	Permitted	Not permitted	Not permitted
FPSCR	Not permitted	Permitted	Not permitted	Permitted
MVFR0, MVFR1	Permitted	Permitted	Not permitted	Not permitted
FPEXC	Permitted	Permitted	Not permitted	Not permitted

## 5.4.2 Accessing the FPU registers

Access to the FPU registers is controlled the CPACR. See [Coprocessor Access Control Register on page 4-27](#) for information on this register.

To use the FPU, you must define the CPACR and *Floating-Point Exception Register* (FPEXC) registers to enable the FPU:

1. Set the CPACR for access to CP10 and CP11 (the FPU coprocessors):

```
LDR r0, =(0xF << 20)
MCR p15, 0, r0, c1, c0, 2
```

2. Set the FPEXC EN bit to enable the FPU:

```
MOV r3, #0x40000000
VMSR FPEXC, r3
```



## 5.5 Register descriptions

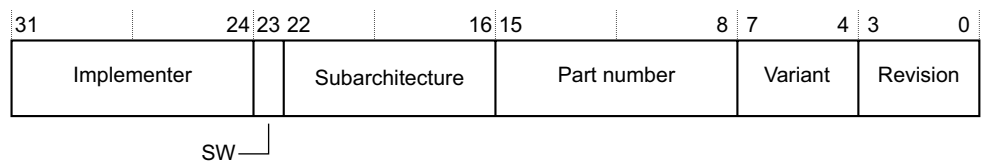
This section describes the FPU system registers. [Table 5-2 on page 5-6](#) provides cross references to individual registers.

### 5.5.1 Floating-Point System ID Register

The FPSID Register characteristics are:

- Purpose** Provides information about the VFP implementation.
- Usage constraints** Only accessible in privileged modes.
- Configurations** Available in all FPU configurations.
- Attributes** See the register summary in [Table 5-2 on page 5-6](#).

[Figure 5-1](#) shows the FPSID Register bit assignments.



**Figure 5-1 FPSID Register bit assignments**

[Table 5-4](#) shows the FPSID Register bit assignments.

**Table 5-4 FPSID Register bit assignments**

Bits	Name	Function
[31:24]	Implementer	Denotes ARM
[23]	SW	Hardware implementation with no software emulation
[22:16]	Subarchitecture	The v2 VFP sub-architecture
[15:8]	Part number	VFPv3
[7:4]	Variant	Cortex-R7
[3:0]	Revision	Revision 0

You can access the FPSID Register with the following VMRS instruction:

VMRS <Rd>, FPSID ; Read Floating-Point System ID Register

### 5.5.2 Floating-Point Status and Control Register

The FPSCR characteristics are:

- Purpose** Provides user-level control of the FPU.
- Usage constraints** There are no usage constraints.
- Configurations** Available in all FPU configurations.
- Attributes** See the register summary in [Table 5-2 on page 5-6](#).

[Figure 5-2 on page 5-9](#) shows the FPSCR bit assignments.

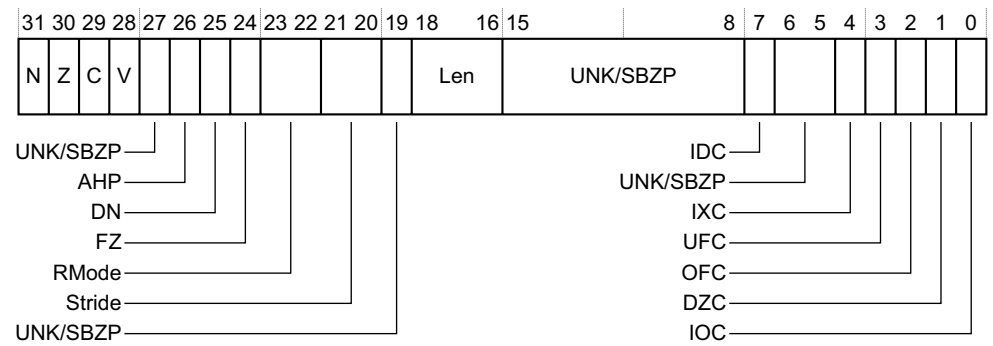


Figure 5-2 FPSCR bit assignments

Table 5-5 shows the FPSCR bit assignments.

Table 5-5 FPSCR bit assignments

Bits	Name	Function
[31]	N	Set to 1 if a comparison operation produces a less than result.
[30]	Z	Set to 1 if a comparison operation produces an equal result.
[29]	C	Set to 1 if a comparison operation produces an equal, greater than, or unordered result.
[28]	V	Set to 1 if a comparison operation produces an unordered result.
[27]	-	UNK/SBZP.
[26]	AHP	Alternative half-precision control bit: 0b0 IEEE half-precision format selected. 0b1 Alternative half-precision.
[25]	DN	Default NaN mode control bit: 0b0 NaN operands propagate through to the output of a floating-point operation. 0b1 Any operation involving one or more NaNs returns the Default NaN. Advanced SIMD arithmetic always uses the Default NaN setting, regardless of the value of the DN bit.
[24]	FZ	Flush-to-zero mode control bit: 0b0 Flush-to-zero mode disabled. Behavior of the floating-point system is fully compliant with the IEEE 754 standard. 0b1 Flush-to-zero mode enabled. Advanced SIMD arithmetic always uses the Flush-to-zero setting, regardless of the value of the FZ bit.
[23:22]	RMode	Rounding Mode control field: 0b00 Round to nearest (RN) mode. 0b01 Round towards plus infinity (RP) mode. 0b10 Round towards minus infinity (RM) mode. 0b11 Round towards zero (RZ) mode. Advanced SIMD arithmetic always uses the Round to nearest setting, regardless of the value of the RMode bits.
[21:20]	Stride	Stride control used for backwards compatibility with short vector values. See the <i>ARM® Architecture Reference Manual ARMv7-A and ARMv7-R edition</i> .
[19]	-	UNK/SBZP.
[18:16]	Len	Vector length, used for backwards compatibility with short vector values. See the <i>ARM® Architecture Reference Manual ARMv7-A and ARMv7-R edition</i> .

### Table 5-5 FPSCR bit assignments (continued)

Bits	Name	Function
[15:8]	-	UNK/SBZP.
[7]	IDC	Input Denormal cumulative exception flag. <sup>a</sup>
[6:5]	-	UNK/SBZP.
[4]	IXC	Inexact cumulative exception flag. <sup>a</sup>
[3]	UFC	Underflow cumulative exception flag. <sup>a</sup>
[2]	OFC	Overflow cumulative exception flag. <sup>a</sup>
[1]	DZC	Division by Zero cumulative exception flag. <sup>a</sup>
[0]	IOC	Invalid Operation cumulative exception flag. <sup>a</sup>

- a. The exception flags, bit [7] and bits [4:0] of the FPSCR are exported on the **FPUFLAGS** output so they can be monitored externally to the processor, if required.

You can access the FPSCR with the following VMSR instructions:

VMRS <Rd>, FPSCR ; Read Floating-Point Status and Control Register  
VMSR FPSCR, <Rt> ; Write Floating-Point Status and Control Register

### 5.5.3 Floating-Point Exception Register

The FPExc Register characteristics are:

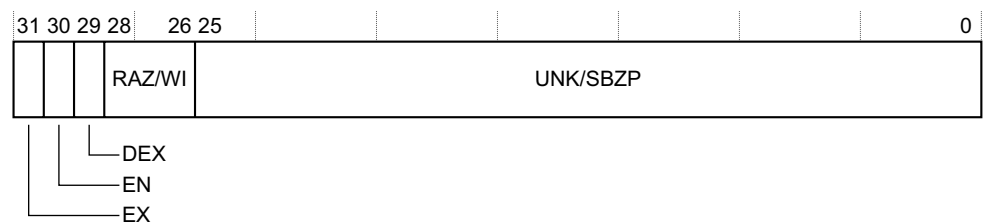
<b>Purpose</b>	Provides global enable control of the Advanced SIMD and VFP extensions.
----------------	---

**Usage constraints** Accessible in all FPU configurations, with restrictions. See *Processor modes for accessing the FPU system registers* on page 5-6.

<b>Configurations</b>	Available in all FPU configurations.
-----------------------	--------------------------------------

**Attributes** See the register summary in [Table 5-2 on page 5-6](#).

Figure 5-3 shows the FPEXC Register bit assignments.



**Figure 5-3 FPEXC Register bit assignments**

Table 5-6 shows the FPEXC Register bit assignments.

**Table 5-6 FPEXC Register bit assignments**

Bits	Name	Function
[31]	EX	Exception bit: This bit reads-as-zero and ignores writes. The Cortex-R7 FPU never requires asynchronous exception handling.
[30]	EN	Enable bit: 0b0 VFP extension is disabled. 0b1 VFP extension is enabled and operates normally. The EN bit is cleared to 0 at reset.
[29]	DEX <sup>a</sup>	Defined synchronous instruction exceptional flag: 0b0 No exception has occurred. 0b1 Attempt to perform a VFP vector operation has been trapped <sup>b</sup> The DEX bit is cleared to 0 at reset.
[28:26]	-	RAZ/WI.
[25:0]	-	UNK/SBZP.

- a. In single-precision only configurations, this bit is not set for any double-precision operations, regardless of whether they are vector operations or not.
- b. The Cortex-R7 FPU hardware does not support the deprecated VFP short vector feature. Attempts to execute VFP data-processing instructions when the FPSCR.LEN field is non-zero result in the FPSCR.DEX bit being set and a synchronous UNDEFINED instruction exception being taken. You can use software to emulate the short vector feature, if required.

You can access the FPEXC Register with the following VMSR instructions:

VMRS <Rd>, FPEXC ; Read Floating-Point Exception Register  
VMSR FPEXC, <Rt> ; Write Floating-Point Exception Register

# Chapter 6

## Level One Memory System

This chapter describes the L1 memory system. It contains the following sections:

- *About the L1 memory system on page 6-2.*
- *Fault handling on page 6-3.*
- *About the TCMs on page 6-7.*
- *About the caches on page 6-8.*
- *Local exclusive monitor on page 6-17.*
- *Memory types and L1 memory system behavior on page 6-18.*
- *Error detection events on page 6-19.*

## 6.1 About the L1 memory system

The processor L1 memory system can be configured during implementation and integration. It consists of:

- Optional separate Harvard instruction and data caches.
- Two optional TCM areas:
  - a configurable *Data TCM* (DTCM) from 0KB to 128KB with no wait state support.
  - a configurable *Instruction TCM* (ITCM) from 0KB to 128KB with optional wait state support.
- An MPU.

---

### Note

---

- If both caches and TCMs are present, instructions can be accessed from both the instruction cache and the ITCM.
  - Instructions cannot be accessed from the DTCM.
- 

Each TCM and cache can be configured at implementation time to have an error detection and correction scheme to protect the data stored in the memory from errors. The TCMs are protected by ECC. [Chapter 7 Fault Detection](#) describes the error detection and correction schemes.

The MPU is unified, and handles accesses to both the instruction and data sides. The MPU is responsible for protection checking, address access permissions, and memory attributes. Some of these functions can be passed to the L2 memory system through the AXI master. See [Memory Protection Unit on page 8-3](#) for more information about the MPU.

The L1 memory system includes a local monitor for exclusive accesses. Exclusive load and store instructions can be used, for example, LDREX, STREX, with the appropriate memory monitoring to provide inter-process or inter-processor synchronization and semaphores. See the *ARM® Architecture Reference Manual ARMv7-A and ARMv7-R edition* for more information.

### 6.1.1 Data cache policy

The following memory regions are cached in L1 data cache:

- Normal memory, Write-Back, non-shareable.
- Normal memory, Write-Back, shareable if multiprocessing is enabled.

---

### Note

---

The Write-Through cache policy is not supported. The *Memory Reconstruction Port* (MRP) provides a way to rebuild step-by-step memory accesses. See [Memory Reconstruction Port on page 2-23](#).

---

## 6.2 Fault handling

Faults can occur on instruction fetches for the following reasons:

- MPU background fault.
- MPU permission fault.
- External AXI3 slave error (SLVERR).
- External AXI3 decode error (DECERR).
- Breakpoints, and vector capture events.

Faults can occur on data accesses for the following reasons:

- MPU background fault.
- MPU permission fault.
- MPU alignment fault.
- External AXI3 slave error (SLVERR).
- External AXI3 decode error (DECERR).
- Watchpoints.

The following sections describe fault handling:

- [Faults](#).
- [Fault status information on page 6-4](#).
- [Usage models on page 6-5](#).

### 6.2.1 Faults

The classes of fault that can occur are:

- [MPU faults](#).
- [External faults](#).
- [Debug events](#).

#### MPU faults

The MPU can generate an abort for various reasons. See [MPU faults on page 8-10](#) for more information. MPU faults are always synchronous, and take priority over other types of abort. If an MPU fault occurs on an access that is not in the TCM, and is non-cacheable, or has generated a cache-miss, the AXI3 transactions for that access are not performed.

#### External faults

A memory access performed through the AXI3 master interface can generate two different types of error response, a slave error (SLVERR) or decode error (DECERR). These are known as external errors, because they are generated by the AXI3 system outside the processor. Synchronous aborts are generated for instruction fetches. All loads and stores generate asynchronous aborts.

#### Debug events

The debug logic in the processor can be configured to generate breakpoints or vector capture events on instruction fetches, and watchpoints on data accesses. If the processor is software-configured for monitor-mode debugging, an abort is taken when one of these events occurs, or when a BKPT instruction is executed. For more information, see [Debug on page 10-12](#).

## Synchronous aborts

An abort is synchronous when the exception is guaranteed to be taken on the instruction that generated the aborting memory access. The abort handler can use the value in the Link Register (r14\_abt) to determine which instruction generated the abort, and the value in the Saved Program Status Register (SPSR\_abt) to determine the state of the processor when the abort occurred.

## Asynchronous aborts

An abort is asynchronous when the exception is taken on an instruction after the instruction that generated the aborting memory access. The abort handler cannot determine which instruction generated the abort or the state of the processor when the abort occurred. Therefore, asynchronous aborts are normally fatal.

All external aborts on both loads and stores to any memory type, that is, normal, device, or strongly-ordered, generate asynchronous aborts on the Cortex-R7 MPCore processor. When the store instruction is committed, the data is normally written into a buffer that holds the data until the memory system has sufficient bandwidth to perform the write access. This gives read accesses higher priority. The write data can be held in the buffer for a long period, during which many other instructions can complete. If an error occurs when the write is finally performed, this generates an asynchronous abort.

### Asynchronous abort masking

The nature of asynchronous aborts means that they can occur while the processor is handling a different abort. If an asynchronous abort generates a new exception in such a situation, the r14\_abt and SPSR\_abt values are overwritten. If this occurs before the data is pushed to the stack in memory, the state information about the first abort is lost. To prevent this from happening, the CPSR contains a mask bit, the A-bit, to indicate that an asynchronous abort cannot be accepted. When the A-bit is set, any asynchronous abort that occurs is held pending by the processor until the A-bit is cleared, when the exception is actually taken. The A-bit is automatically set when abort, IRQ or FIQ exceptions are taken, and on reset. You must only clear the A-bit in an abort handler after the state information has either been stacked to memory, or is no longer required.

The processor supports only one pending asynchronous external abort. If a subsequent asynchronous external abort is signaled while another one is pending, the later one is ignored and only one abort is taken.

### Memory barriers

When a store instruction, or series of instructions has been executed, it is sometimes necessary to determine whether any errors occurred because of these instructions. Because most of these errors are reported asynchronously, they might not generate an abort exception until some time after the instructions are executed. To ensure that all possible errors have been reported, you must execute a DSB instruction. If the CPSR A-bit is clear, these errors are not masked and the abort exceptions are taken. If the A-bit is set, the aborts are held pending.

## 6.2.2 Fault status information

When an abort occurs, information about the cause of the fault is recorded in a number of registers, depending on the type of abort:

- [Abort exceptions on page 6-5.](#)
- [Synchronous abort exceptions on page 6-5.](#)



## Abort exceptions

The following registers are updated when any abort exception is taken:

### Link Register

The `r14_abt` register is updated to provide information about the address of the instruction that the exception was taken on, in a similar way to other types of exception. This information can be used to resume program execution after the abort has been handled.

#### Note

When a prefetch abort has occurred, ARM recommends that you do not use the link register value for determining the aborting address, because 32-bit Thumb instructions do not have to be word aligned and can cause an abort on either halfword. This applies even if all of the code in the system does not use the extra 32-bit Thumb instructions introduced in ARMv6T2, because the earlier BL and BLX instructions are both 32 bits long.

### Saved Program Status Register

The `SPSR_abt` register is updated to record the state and mode of the processor when the exception was taken, in a similar way to other types of exception.

### Fault Status Register

There are two fault status registers, one for prefetch aborts (IFSR) and one for data aborts (DFSR). These record the type of abort that occurred, and whether it occurred on a read or a write. In particular, this enables the abort handler to distinguish between synchronous aborts, asynchronous aborts, and debug events.

## Synchronous abort exceptions

The following register is updated when a synchronous abort exception is taken:

### Fault Address Register

There are two fault address registers, one for prefetch aborts (IFAR) and one for data aborts (DFAR). These indicate the address of the memory access that caused the fault.

## 6.2.3 Usage models

This section describes some ways to handle errors in a system. Exactly how you program the Cortex-R7 MPCore processor to handle errors depends on the configuration of your processor and system, and what you are trying to achieve.

If an abort exception is taken, the abort handler reads the information in the link register, `SPSR`, and fault status registers to determine the type of abort. Some types of abort are fatal to the system, and others can be fixed, and program execution resumed. For example, an MPU background fault might indicate a stack overflow, and be rectified by allocating more stack and reprogramming the MPU to reflect this. Alternatively, an asynchronous external abort might indicate that a software error meant that a store instruction occurred to an unmapped memory address. This type of abort is fatal to the system or process because no information is recorded about the address the error occurred on, or the instruction that caused the error.

[Table 6-1 on page 6-6](#) shows which types of abort are typically fatal because either the location of the error is not recorded or the error is unrecoverable. Some aborts that are marked as not fatal might be fatal in some systems when the cause of the error is determined. For example, an MPU background fault might indicate something that can be rectified, for example a stack overflow.

It might also indicate something that is fatal, for example that because of a bug, the software has accessed a nonexistent memory location. These cases can be distinguished by determining the location where the error occurred. If an error is unrecoverable, it is normally fatal regardless of whether or not the location of the error is recorded.

**Table 6-1 Types of aborts**

Type	Conditions	Source	Synchronous	Fatal
MPU fault	Access not permitted by MPU <sup>a</sup>	MPU	Yes	No
Asynchronous external	Any external access	AXI3	No	Yes

a. See [MPU faults on page 8-10](#) for more information about the types of MPU fault.

### Correctable errors

You can configure the processor to respond to and automatically correct ECC errors. Connect the event output or outputs that indicate a correctable error to an interrupt controller. When such an event occurs, the interrupt input to the processor is set, and the processor takes an interrupt exception. When your interrupt handler has identified the source of the interrupt as a correctable error, it can read the [ECC Error Registers on page 4-39](#) to determine where the ECC error occurred. You can examine this information to identify trends in such errors. By masking the interrupt when necessary, your software can ensure that when critical code is executing, the processor corrects the errors automatically, but delays examining information about the error until after the critical code has completed.

When the processor is in debug state, any correctable error is corrected. However, in case of load instruction, the memory access replay to read the corrected data is not done, and therefore the instruction generating the error does not complete successfully. Instead, the sticky synchronous abort flag in the DBGDSCR is set.

### Debugging cache and TCM access

See [Cache and TCM Debug Operation Register on page 4-36](#).

## 6.3 About the TCMs

See [Instruction and data TCM on page 8-15](#) for information on the TCMs.

## 6.4 About the caches

The L1 memory system can be configured to include instruction and data caches of varying sizes. You can configure the size of each cache independently. The cached instructions or data are fetched from external memory using the L2 memory interface. The cache controllers use the RAMs that are integrated into the Cortex-R7 MPCore macrocell during implementation.

If an access is to Cacheable memory, and the cache is enabled, a lookup is performed in the cache and, if found in the cache, that is, a cache hit, the data is fetched from or written into the cache. When the cache is not enabled and for Non-cacheable memory, the accesses are performed using the L2 memory interface.

The cache controllers also manage the cache maintenance operations described in [Cache maintenance operations](#).

Each cache can also be configured with ECC error checking schemes. If an error checking scheme is implemented and enabled, then the tags associated with each line, and data read from the cache are checked whenever a lookup is performed in the cache. See [Cache error detection and correction](#) for more information.

For more information on the general rules about memory attributes and behavior, see the *ARM® Architecture Reference Manual ARMv7-A and ARMv7-R edition*.

### 6.4.1 Cache maintenance operations

All cache maintenance operations are done through the system control coprocessor, CP15. The system control coprocessor operations supported for the data cache are:

- Invalidate by address (MVA).
- Invalidate by Set/Way combination.
- Clean by address (MVA).
- Clean by Set/Way combination.
- Clean and Invalidate by address (MVA).
- Clean and Invalidate by Set/Way combination.
- *Data Memory Barrier (DMB) and Data Synchronization Barrier (DSB) operations.*

The system control coprocessor operations supported for the instruction cache are:

- Invalidate all.
- Invalidate by address.

### 6.4.2 Cache error detection and correction

The processor can detect, handle, report, and correct cache memory errors. This section describes:

- [Error build options on page 6-9.](#)
- [Address decoder faults on page 6-9.](#)
- [Handling cache ECC errors on page 6-9.](#)
- [Errors on instruction cache read on page 6-9.](#)
- [Errors on evictions on page 6-10.](#)
- [Errors on cache maintenance operations on page 6-10.](#)

## Error build options

The caches can detect and correct errors depending on the build options used in the implementation. If the ECC build option is enabled:

- The instruction cache is protected by a 64-bit ECC scheme. The data RAMs include eight bits of ECC code for every 64 bits of data. The tag RAMs include seven bits of ECC code to cover the tag and valid bit.
- The data cache is protected by a 32-bit ECC scheme. The data RAMs include seven bits of ECC code for every 32 bits of data. The tag RAMs include seven bits of ECC code to cover the tag and control bits.

## Address decoder faults

The error detection schemes described in this section provide protection against errors that occur in the data stored in the cache RAMs. Each RAM normally includes a decoder that enables access to that data and, if an error occurs in this logic, it is not normally detected by these error detection schemes. The processor includes features that enable it to detect some address decoder faults.

## Handling cache ECC errors

Table 6-2 shows the behavior of the processor on a cache ECC error, depending on bit[9] of the ACTLR.

**Table 6-2 Cache ECC error behavior**

Bit	Behavior
[9]	ECC on.

See [Disabling or enabling error checking on page 6-16](#) for information on how to safely change these bits.

When ECC checking is enabled, hardware recovery is always enabled. When an ECC error is detected, the processor tries to evict the cache line containing the error. If the line is clean, it is invalidated, and the correct data is reloaded from the L2 memory system. If the line is dirty, the eviction writes the dirty data out to the L2 memory system, and in the process it corrects any 1-bit errors. The corrected data is then reloaded from the L2 memory system.

If a 2-bit error is detected in data ram for a dirty line or in tag RAM, the error is not correctable. If the 2-bit error is in the tag RAM, no data is written to the L2 memory system. If the 2-bit error is in the data RAM, the cache line is written to the L2 memory system, but the AXI master port **WSTRBM** signal is LOW for the data that contains the error. If an uncorrectable error is detected, an ECC primary output is always generated because data might have been lost. It is expected that such a situation can be fatal to the software process running.

## Errors on instruction cache read

All ECC errors detected on instruction cache reads are correctable.

All detectable errors in the instruction cache can always be recovered from because the instruction cache never contains dirty data.

## Errors on evictions

If the cache controller has determined a cache miss has occurred, it might have to do an eviction before a linefill can take place. This can occur on reads and writes. Certain cache maintenance operations also generate evictions. If it is a data-cache line that is dirty, an ECC error might be detected on the line being evicted:

- If the error is correctable, it is corrected inline before the data is written to the external memory using the L2 memory interface.
- If there is an uncorrectable error in the tag RAM, the write is not done.
- If there is an uncorrectable error in the data RAM, the AXI master port **WSTRBM** signal is deasserted for the word(s) with an error.

## Errors on cache maintenance operations

The following sections describe errors on cache maintenance operations:

- *Invalidate all instruction cache.*
- *Invalidate instruction cache by address.*
- *Invalidate data cache by address.*
- *Invalidate data cache by set/way.*
- *Clean data cache by address.*
- *Clean data cache by set/way on page 6-11.*
- *Clean and invalidate data cache by address on page 6-11.*
- *Clean and invalidate data cache by set/way on page 6-11.*

### ***Invalidate all instruction cache***

This operation does not generate any errors.

### ***Invalidate instruction cache by address***

This operation requires a cache lookup. Any errors found in the set that was looked up are fixed by invalidating that line and, if the address in question is found in the set, it is invalidated.

Any detected error is signaled with the appropriate event.

### ***Invalidate data cache by address***

This operation requires a cache lookup. Any correctable errors found in the set that was looked up are fixed and, if the address in question is found in the set, it is invalidated.

Any detected error is signaled with the appropriate event.

### ***Invalidate data cache by set/way***

This operation does not require a cache lookup. It refers to a particular cache line.

### ***Clean data cache by address***

This operation requires a cache lookup. Any correctable errors found in the set that was looked up are fixed and, if the address in question is found in the set, the instruction carries on with the clean operation.

If the tag RAM has an uncorrectable error, the data is not written to memory.

If the line is dirty, the data is written back to external memory. If the data has an uncorrectable error, the words with the error have their **WSTRBM** AXI signal deasserted. If there is a correctable error, the line has the error corrected inline before it is written back to memory.

Any detected error is signaled with the appropriate event.

#### ***Clean data cache by set/way***

This operation does not require a cache lookup. It refers to a particular cache line.

The tag RAMs for the cache line are checked.

If the tag RAM has an uncorrectable error, the data is not written to memory.

If the line is dirty, the data is written back to external memory. If the data has an uncorrectable error, the words with the error have their **WSTRBM** AXI signal deasserted. If there is a correctable error, the line has the error corrected inline before it is written back to memory.

Any detected error is signaled with the appropriate event.

#### ***Clean and invalidate data cache by address***

This operation requires a cache lookup. Any correctable errors found in the set that was looked up are fixed and, if the address in question is found in the set, the instruction carries on with the clean and invalidate operation.

If the tag RAM has an uncorrectable error, the data is not written to memory.

If the line is dirty, the data is written back to external memory. If the data has an uncorrectable error, the words with the error have their **WSTRBM** AXI signal deasserted. If there is a correctable error, the line has the error corrected inline before it is written back to memory.

Any detected error is signaled with the appropriate event.

#### ***Clean and invalidate data cache by set/way***

This operation does not require a cache lookup. It refers to a particular cache line.

The tag RAMs for the cache line are checked.

If the tag RAM has an uncorrectable error, the data is not written to memory.

If the line is dirty, the data is written back to external memory. If the data has an uncorrectable error, the words with the error have their **WSTRBM** AXI signal deasserted. If there is a correctable error, the line has the error corrected inline before it is written back to memory.

Any detected error is signaled with the appropriate event.

### **6.4.3 Data cache RAM organization**

This section describes the data cache RAM organization in the following sections:

- [Tag RAM](#).
- [Data RAM on page 6-13](#).

#### **Tag RAM**

The tag RAMs consist of four ways of up to 512 lines. The width of the RAM depends on the build options selected, and the size of the cache. The following tables show the tag RAM bits:

- [Table 6-3 on page 6-12](#) shows the tag RAM bits when ECC is implemented

- [Table 6-4](#) shows the tag RAM bits when ECC is not implemented.

**Table 6-3 Tag RAM bit descriptions, with ECC**

Bit in tag cache line	Description
Bits[35:29]	ECC code bits
Bits[28:27]	Outer attribute
Bit[26]	Inner allocate attribute
Bit[25]	Dirty bit
Bit[24]	Shareable bit
Bit[23]	Exclusive bit
Bit[22]	Valid bit
Bits[21:0]	Tag value

**Table 6-4 Tag RAM bit descriptions, no ECC**

Bit in tag cache line	Description
Bits[28:27]	Outer attribute
Bit[26]	Inner allocate attribute
Bit[25]	Dirty bit
Bit[24]	Shareable bit
Bit[23]	Exclusive bit
Bit[22]	Valid bit
Bits[21:0]	Tag value

A cache line is marked as valid by bit [22] of the tag RAM. Each valid bit is associated with a whole cache line, so evictions always occur on the entire line.

[Table 6-5](#) shows the tag RAM cache sizes and associated RAM organization, assuming no ECC. For ECC, the width of the tag RAMs must be increased by seven bits.

**Table 6-5 Cache sizes and tag RAM organization**

Cache size	Tag RAM organization
4KB	4 banks 29 bits 32 lines
8KB	4 banks 28 bits 64 lines
16KB	4 banks 27 bits 128 lines
32KB	4 banks 26 bits 256 lines
64KB	4 banks 25 bits 512 lines

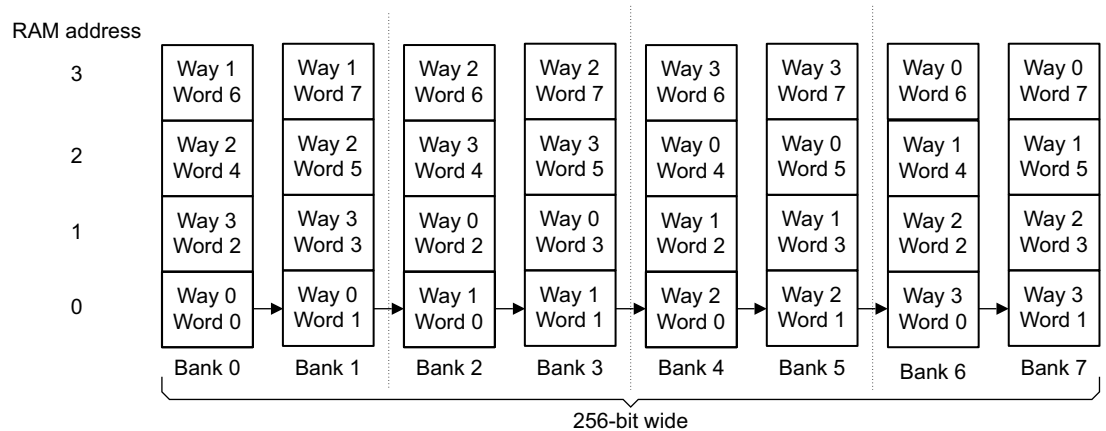


## Data RAM

Data RAM is organized as eight banks of 32-bit wide lines, or in the instruction cache as four banks of 64-bit wide lines. This RAM organization means that it is possible to:

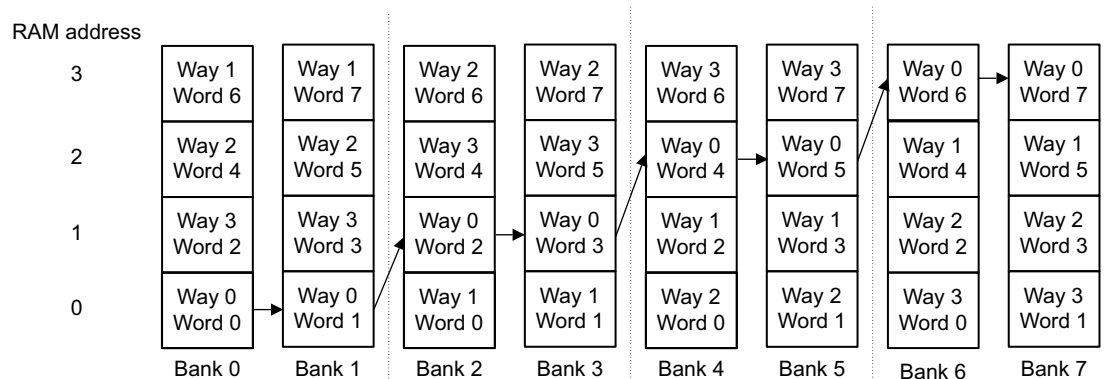
- Perform a cache look-up with one RAM access, all banks selected together. This is done for nonsequential read operations, as shown in [Figure 6-1](#).
- Select the appropriate bank RAM for sequential read operations, as shown in [Figure 6-2](#).
- Write a line to the eviction buffer in one cycle, a 256-bit read access.
- Fill a line in one cycle from the linefill buffer, a 256-bit write access.

[Figure 6-1](#) shows a cache look-up being performed on all banks with one RAM access.



**Figure 6-1 Nonsequential read operation performed with one RAM access.**

[Figure 6-2](#) shows the appropriate bank RAM being selected for a sequential read operation.



**Figure 6-2 Sequential read operation performed with one RAM access**

The data RAM organization is optimized for 64-bit read operations, because with the same address, two words on the same way can be selected.

Data RAM sizes depend on the build option selected, and are described in:

- [Data RAM sizes without ECC implemented on page 6-14.](#)
- [Data RAM sizes with ECC implemented on page 6-14.](#)

**Data RAM sizes without ECC implemented**

Table 6-6 shows the organization for instruction caches when ECC is not implemented.

**Table 6-6 Instruction cache data RAM sizes, no ECC**

Cache size	Data RAMs
4KB, 4 1KB ways	4 banks 64 bits 128 lines or 8 banks 32 bits 128 lines
8KB, 4 2KB ways	4 banks 64 bits 256 lines or 8 banks 32 bits 256 lines
16KB, 4 4KB ways	4 banks 64 bits 512 lines or 8 banks 32 bits 512 lines
32KB, 4 8KB ways	4 banks 64 bits 1024 lines or 8 banks 32 bits 1024 lines
64KB, 4 16KB ways	4 banks 64 bits 2048 lines or 8 banks 32 bits 2048 lines

Table 6-7 shows the organization for data caches when ECC is not implemented.

**Table 6-7 Data cache data RAM sizes, no ECC**

Cache size	Data RAMs
4KB, 4 1KB ways	8 banks 32 bits 128 lines
8KB, 4 2KB ways	8 banks 32 bits 256 lines
16KB, 4 4KB ways	8 banks 32 bits 512 lines
32KB, 4 8KB ways	8 banks 32 bits 1024 lines
64KB, 4 16KB ways	8 banks 32 bits 2048 lines

**Data RAM sizes with ECC implemented**

Table 6-8 shows the organization for the instruction cache when ECC is implemented. ECC error detection adds eight bits for every 64 bits, so four bits are added for each RAM bank.

**Table 6-8 Instruction cache data RAM sizes with ECC**

Cache size	Data RAMs
4KB, 4 1KB ways	4 banks 72 bits 128 lines or 8 banks 36 bits 128 lines
8KB, 4 2KB ways	4 banks 72 bits 256 lines or 8 banks 36 bits 256 lines
16KB, 4 4KB ways	4 banks 72 bits 512 lines or 8 banks 36 bits 512 lines
32KB, 4 8KB ways	4 banks 72 bits 1024 lines or 8 banks 36 bits 1024 lines
64KB, 4 16KB ways	4 banks 72 bits 2048 lines or 8 banks 36 bits 2048 lines

Table 6-9 shows the organization for the data cache when ECC is implemented. ECC error detection adds seven bits for every 32 bits, so seven bits are added for each RAM bank.

**Table 6-9 Data cache data RAM sizes with ECC**

Cache size	Data RAMs
4KB, 4 1KB ways	8 banks 39 bits 128 lines
8KB, 4 2KB ways	8 banks 39 bits 256 lines
16KB, 4 4KB ways	8 banks 39 bits 512 lines
32KB, 4 8KB ways	8 banks 39 bits 1024 lines
64KB, 4 16KB ways	8 banks 39 bits 2048 lines

Table 6-10 shows the organization of the data cache RAM bits when ECC is implemented.

**Table 6-10 Data cache RAM bits, with ECC**

RAM bits	Description
Bits[39:32]	ECC code bits for data [31:0]
Bits[31:0]	Data [31:0]

#### 6.4.4 Cache interaction with memory system

This section describes how to enable or disable the cache RAMs, and to enable or disable error checking. After you enable or disable the instruction cache, you must issue an ISB instruction to flush the pipeline. This ensures that all subsequent instruction fetches see the effect of enabling or disabling the instruction cache.

After reset, you must invalidate each cache before enabling it.

When disabling the data cache, you must clean the entire cache to ensure that any dirty data is flushed to L2 memory.

Before enabling the data cache, you must invalidate the entire data cache if L2 memory might have changed since the cache was disabled.

Before enabling the instruction cache, you must invalidate the entire instruction cache if L2 memory might have changed since the cache was disabled.

##### Disabling or enabling instruction cache

The following code is an example of enabling the instruction cache:

```
MRC p15, 0, R1, c1, c0, 0 ; Read System Control Register configuration data
ORR R1, R1, #0x1 <<12    ; instruction cache enable
MCR p15, 0, r0, c7, c5, 0 ; Invalidate entire instruction cache
MCR p15, 0, R1, c1, c0, 0 ; enabled instruction cache
ISB
```

The following code is an example of disabling the instruction cache:

```
MRC p15, 0, R1, c1, c0, 0 ; Read System Control Register configuration data
BIC R1, R1, #0x1 <<12    ; instruction cache enable
MCR p15, 0, R1, c1, c0, 0 ; disabled instruction cache
ISB
```

## Disabling or enabling data cache

The following code is an example of enabling the data cache:

```
MRC p15, 0, R1, c1, c0, 0 ; Read System Control Register configuration data
ORR R1, R1, #0x1 <<2
DSB
; Invalidate the data cache with a loop of invalidate by set/way operations. This
; routine will depend on the data cache size.
MCR p15, 0, R1, c1, c0, 0 ; enabled data cache
```

The following code is an example of disabling the cache RAMs:

```
MRC p15, 0, R1, c1, c0, 0 ; Read System Control Register configuration data
BIC R1, R1, #0x1 <<2
DSB
MCR p15, 0, R1, c1, c0, 0 ; disabled data cache
; Clean entire data cache. This routine will depend on the data cache size.
```

## Disabling or enabling error checking

The following code is the recommended sequence to perform the change:

```
MRC p15, 0, r0, c1, c0, 0 ; Read System Control Register
BIC r0, r0, #0x1 << 2 ; Disable data cache bit
BIC r0, r0, #0x1 << 12 ; Disable instruction cache bit
DSB
MCR p15, 0, r0, c1, c0, 0 ; Write System Control Register
ISB ; Ensures following instructions are not executed from cache
; Clean entire data cache. This routine will depend on the data cache size. It can be
; omitted if the cache has not been enabled yet.
MRC p15, 0, r1, c1, c0, 1 ; Read Auxiliary Control Register
; Change bits 10:9 as needed
MCR p15, 0, r1, c1, c0, 1 ; Write Auxiliary Control Register
; Invalidate the data cache. This routine will depend on the data cache size
MCR p15, 0, r0, c7, c5, 0 ; Invalidate entire instruction cache
MRC p15, 0, r0, c1, c0, 0 ; Read System Control Register
ORR r0, r0, #0x1 << 2 ; Enable data cache bit
ORR r0, r0, #0x1 << 12 ; Enable instruction cache bit
DSB
MCR p15, 0, r0, c1, c0, 0 ; Write System Control Register
ISB
```

## 6.5 Local exclusive monitor

The processor L1 memory system has an local exclusive monitor. This is a two state, open and exclusive, state machine that manages load/store exclusive (LDREXB, LDREXH, LDREX, LDREXD, STREXB, STREXH, STREX and STREXD) accesses and clear exclusive (CLREX) instructions. You can use these instructions, operating in the L1 memory system, to construct semaphores and ensure synchronization between different processes. By adding a global exclusive monitor, you can also use these instructions in the L2 memory system to construct semaphores and ensure synchronization between different processors. See [Chapter 11 Level Two Interface](#). See the *ARM® Architecture Reference Manual ARMv7-A and ARMv7-R edition* for more information about how these instructions work.

When a load-exclusive access is performed, the local exclusive monitor moves to the exclusive state. It can also move back to the open state for other reasons, for example, the other processor has taken the semaphore, or because of eviction of the cache line containing the semaphore value. The local exclusive monitor holds exclusivity state for the Cortex-R7 MPCore processor only. It does not record the address of the memory that a load-exclusive access was performed to. Any store exclusive access performed when the state is open fails.

## 6.6 Memory types and L1 memory system behavior

The behavior of the L1 memory system depends on the type attribute of the memory that is being accessed:

- Only Normal, Write Back memory can be cached in the RAMs.
- Normal, Write Through memory is treated as non-cacheable.
- Normal, Write Back, Shareable memory is treated as cacheable when ACTLR.SMP is set to 1.
- Normal, Write Back, Non-shareable memory is always treated as cacheable.
- Only Normal memory is considered restartable. A multi-word transfer can be abandoned part way through because of an interrupt, and be restarted after the interrupt has been handled.
- Only the local exclusive monitor is used for exclusive accesses to Non-shareable Write Back memory and to coherent cacheable shareable memory. Other exclusive accesses are checked using the local monitor and also, if necessary, any global monitor, using the L2 memory interface.
- Exclusive accesses to the ITCM fail.
- Normal non-cacheable exclusive accesses are handled using both local and global monitor.
- Accesses resulting from SWP and SWPB instructions to Cacheable memory are not marked as locked when performed using the L2 memory interface.

Table 6-11 summarizes the processor memory types and associated behavior.

**Table 6-11 Memory types and associated behavior**

Memory type		Cacheable	Merging	Restartable	Local exclusives	Locked swaps
Normal	Shareable	— <sup>a</sup>	Yes	Yes	Partially <sup>b</sup>	Yes
	Non-shareable	Yes	Yes	Yes	Yes	No
Device	Shareable	No	No	No	No <sup>c</sup>	No <sup>c</sup>
	Non-shareable	No	No	No	No <sup>c</sup>	Yes
Strongly Ordered	Shareable	No	No	No	No <sup>c</sup>	Yes

a. Depends on the value of the ACTLR.SMP bit:

1 = Cacheable.

0 = Non-cacheable.

b. Depends on the value of the ACTLR.SMP bit:

1 = Exclusive accesses handled using only local monitor.

0 = Exclusive accesses handled using both local and global monitor.

c. Exclusive accesses are handled using both local and global monitor.

## 6.7 Error detection events

See [Performance monitoring events on page 10-7](#) for information on error detection events.

# Chapter 7

## Fault Detection

This chapter describes the fault detection features of the Cortex-R7 MPCore processor. It contains the following sections:

- *About fault detection on page 7-2.*
- *RAM protection on page 7-3.*
- *Logic protection on page 7-10.*
- *External memory and bus protection on page 7-11.*
- *Programmers view on page 7-12.*
- *ECC on RAMs on page 7-5.*
- *ECC on external AXI bus on page 7-7.*
- *Lock-step on page 7-14.*
- *Static split/lock on page 7-16.*



## 7.1 About fault detection

The Cortex-R7 MPCore processor fault detection features:

- Report any detected error to the system.
- Correct any detected and correctable error.

When the system-on-chip reports a failure, the logging and status must provide all the required information to identify the source of the failure. The intention is to detect and correct the errors as much as possible, and identify them to the system.

For *Error Correcting Code* (ECC) on RAM, all errors explicitly seen must be reported, so that the error propagation is minimized.

### 7.1.1 RAM and logic protection

The processor protects memories and logic in two different ways. These are independent, and can be used separately.

- The RAMs are protected with ECC, except the branch prediction RAMs, that only have parity protection.
- The logic of the individual processors is protected by duplication with diagnostic compare. This is known as redundant logic in a lock-step or split/lock implementation. The SCU logic is also duplicated. The AXI buses can also be protected with ECC and parity.

These are independent, and can be used separately.

The processor uses *Single Error Correction* and *Double Error Detection* (SEC-DED) ECCs to detect and correct errors in the RAMs and on the AXI buses. A finite number of hard, that is, permanent errors can be detected and corrected with continued operation using dedicated error registers.

### 7.1.2 Analysis of errors

A monitor external to the processor is responsible for analyzing the notified error and marking the corrupted entries as reusable if it has been proven to be a soft error. This analysis can be performed as follows:

- Through the MBIST port when setting the processors of the cluster to WFI mode.
- By CP15/SCU direct cache access registers. These registers enable limited access to the caches and SCU tag RAM duplicates.

---

#### **Note**

MBIST routines generated by a controller external to the device can be used to analyze the full RAMs at boot or on demand. CP15 and SCU direct accesses can analyze a particular location.

---

## 7.2 RAM protection

The following sections describe RAM protection:

- [Protection method](#).
- [RAM protection summary on page 7-5](#).
- [ECC codes on page 7-8](#).
- [RAM configuration on page 7-8](#).
- [Performance impact on page 7-9](#).

### 7.2.1 Protection method

The following sections describe how RAM errors are managed:

- [Detecting errors](#).
- [Correcting errors](#).
- [Handling permanent errors on page 7-4](#).
- [Reporting errors on page 7-4](#).

#### Detecting errors

The Cortex-R7 MPCore processor uses ECC to indicate errors on the RAMs. ECC can also correct errors because the probability of single errors is much higher than the probability of multiple errors in the same ECC chunk. This is done by arranging the RAMs so that physically contiguous positions in the RAMs do not correspond to the same ECC chunks.

#### Correcting errors

The Cortex-R7 MPCore processor implements RAM error correction using a clean and invalidate and retry for caches, and a correct, writeback, and retry mechanism for TCMs. When a correctable error is detected, as shown in [Table 7-1 on page 7-5](#), the corresponding index/way is cleaned and invalidated. When the clean and invalidate operation completed, the requester retries its access.

#### ————— Note —————

The detection of multiple-bit errors is not synchronous. Therefore, when such an error is notified, corrupted data might not be contained. Contact ARM for more details about multiple-bit ECC errors.

#### Instruction side

On the instruction side, lines are always clean so that invalidating the line is sufficient. The retried access then fetches the correct value from the upper level memory.

#### Data side

On the data side, the cache line can be dirty. The correction of the read contents is done as part of the clean and invalidate operation for caches. This takes place in the eviction buffer and in the cache coherency block. For TCMs, correction of the read contents is done with a correct and writeback operation.

#### SCU

The detection of an error in the duplicate of the tags of a processor causes a clean and invalidate in the corresponding processor tag RAM. When the clean and invalidate is done, the line in the SCU tag RAM is marked as unusable.

## Handling permanent errors

Permanent errors are handled as follows:

### General behavior

If hard, or permanent, errors occur on the RAMs, the clean/invalidate and retry scheme might cause a deadlock, and the access is continuously replayed. To prevent this, error bank registers are provided to mask the faulty locations as unusable and invalid. When an error is detected, the location is pushed in the bank that masks the corresponding valid bit of the location when reading and when allocating a new line. The line is therefore no longer used unless the entry is reset by a CP15 access. There is a short period of time during which the line is still seen by the system, but is removed from the allocation pool.

The depth of the error bank determines how many errors can be supported by the system. When this limit is reached, the system might livelock. The processor provides a special ECC event indicating the number of corrupted location to monitor the error bank status before it becomes full. This is a condition that can cause a potential deadlock. This information is reported on several pins signaling the usage of the error bank, that is, showing if the error bank is empty or at least one error has been encountered. See [Error detection notification signals on page 7-13](#).

### Interaction between SCU and processors

For a processor error, the line is cleaned and invalidated and the ECC error bank prevents any future allocation in this way. However, the line is still seen as present by the SCU, and the SCU requests the line to the processor that misses or hits, depending on whether the line has been reallocated in another cache location.

For an SCU error, the line is marked as unusable by the SCU error bank but the processor still sees the line as usable. Therefore, a processor can request an access to this way to allocate the cache line, but the write fails in the SCU without being reported. Because of this, the error seen by SCU is sent back to the processor, and stored in the processor data error bank.

### Reporting errors

The Cortex-R7 MPCore processor notifies the detection of any error using primary output events, and the update of performance and statistics counters.

### 7.2.2 RAM protection summary

Table 7-1 shows how the different types of RAM are protected.

**Table 7-1 RAM protection summary**

RAM type	Protection	Parity/ECC chunk	Correctable error	Fatal error	Hard error support
Data tag RAM	SEC-DED ECC	34 bits: <ul style="list-style-type: none"> <li>25 bits for tag and status.</li> <li>9 bits for index.</li> </ul>	Error seen as a single bit error	Error seen as a multiple bit error	Up to three hard errors, including the SCU hard errors
Data data RAM	SEC-DED ECC	32-bit data word	Error seen as a single bit error	Error seen as a multiple bit error on dirty lines	
Instruction tag RAM	SEC-DED ECC	28 bits: <ul style="list-style-type: none"> <li>23 bits for tag.</li> <li>5 bits for index.</li> </ul>	Any error, single or double, on the tag or valid stored in the RAM	None	Up to three hard errors
Instruction data RAM	SEC-DED ECC <sup>a</sup>	64-bit data word	Any error on the data stored in the RAM	None	
BTAC	Parity <sup>b</sup>	8-bit	-	-	None
PRED	Parity <sup>b</sup>	1-bit (copy)	-	-	None
SCU tag RAM	SEC-DED ECC	28 bits: <ul style="list-style-type: none"> <li>23 bits for tag.</li> <li>5 bits for index.</li> </ul>	Any error	None	Up to two hard errors
Data TCM	SEC-DED ECC	32-bit data word	Error seen as a single bit error	Error seen as a multiple bit error	Up to one hard error
Instruction TCM	SEC-DED ECC	64-bit word	Error seen as a single bit error	Error seen as a multiple bit error	Up to one hard error

a. The SEC-DED ECC is used as a Double Error Correction because the lines are clean.

b. BTAC and PRED do not prevent the system from operating correctly and only impact the performance, even if hard errors occur. Parity provides a compromise between the area overhead in the RAMs and the ability to detect errors.

### 7.2.3 ECC on RAMs

This section describes the use of ECC on the RAMs in:

- [RAM targeted](#).
- [Basic scheme on page 7-6](#).
- [Auto-check mechanism on page 7-7](#).
- [MBIST for full RAM analysis on page 7-7](#).

#### RAM targeted

To prevent the loss of any data that might cause the processor or the system to malfunction, the ECC protects the following:

- L1 data cache RAMs.
- L1 instruction cache RAMs.
- SCU Tag RAM.
- Instruction TCM.
- Data TCM.

For more information on the RAM targeted and the level of fault detection support, see [RAM protection summary on page 7-5](#).

### Basic scheme

The basic ECC scheme for the L1 cache, SCU RAM, and TCM is as follows:

- Any error, either soft or hard, is indicated to the system.
- Any error detected is stored in an error bank, waiting to be analyzed by the system, and preventing that location from being used.
- The system analyzes the faulty RAM location to check the full RAM space (MBIST). See [Analysis of errors on page 7-2](#).
- The processor can also analyze the faulty RAM location to check a single location or a range of locations (CP15 operation).
- From the analysis, the errors detected are classified as soft or hard errors. If the errors are hard, the processor updates the error bank with this information and the corresponding RAM location is not used by subsequent accesses. There is one error bank for each of the following:
  - L1 data cache RAMs.
  - L1 instruction cache RAMs.
  - SCU Tag RAM.
  - Instruction TCM.
  - Data TCM.

[Table 7-2](#) shows the basic scheme, and also the differences where it is part of the processor, that is, L1 cache and TCM, or is seen as a peripheral such as the SCU.

**Table 7-2 Basic ECC scheme per RAM type**

Description	L1 cache	SCU	TCM
Correctable error notification to the system, bits[10:9] of the ACTLR cleared	-	-	-
Correctable error notification to the system, bits[10:9] of the ACTLR set	Error notification <sup>a</sup>	-	Error notification <sup>a</sup>
Uncorrectable error notification to the system	Error notification <sup>a</sup>	Error notification <sup>a</sup>	Error notification <sup>a</sup>
Logging of errors	Up to three entries in the error bank	Up to two entries in the error bank	One entry in the error bank
Direct access to the faulty RAM location by the system for full RAM analysis	MBIST	MBIST	MBIST or slave port
Direct access to the faulty RAM location by the processor itself for single location analysis	CP15 Debug Cache Access Registers	Memory-mapped register in the SCU	CP15 Debug TCM Access Registers
Access to the error bank to update it after analysis of the faulty RAM location	CP15	Memory-mapped register in the SCU	CP15

a. Fault detection error notification is done by the primary output pins. See [Error detection notification signals on page A-25](#).

### Auto-check mechanism

The direct access to the faulty RAM location by the processor for single location analysis also enables errors to be injected so that the error handling mechanism can be checked. This provides a full software support to generate errors on particular locations in the RAM, and then enabling and disabling the ECC after those accesses. See [SCU Debug tag RAM access on page 9-16](#) and [Cache and TCM Debug Operation Register on page 4-36](#) for more information.

### MBIST for full RAM analysis

The MBIST interface can be used during WFI. The MBIST controller has some arbitration on the RAMs and can get a clock running in the processor clock module when the **MBISTENABLE** signal is active, so that flops before and after the RAMs can be activated. Any other MBIST usage while the processor is running is not supported.

#### 7.2.4 ECC on external AXI bus

All master buses, that is, AXI master port 0 and AXI master port 1, and the peripheral port, generate ECC check bits that are computed on all signals of the bus for write accesses, such as payload and control, including the AXI valid and AXI ready signals.

For read accesses, the complete bus is decoded:

- For single-bit errors, an inline correction is provided. A primary error detection notification output signal is raised at the same time. The inline correction implies some extra cycles of memory latency.  
Inline correction is not done on the TCM AXI slave port.
- Double-bit errors only raise a primary error detection notification output signal. The system must determine the correct action in this case, such as an interrupt to the processor.

An external write access must be decoded on the ACP:

- A single-bit error is corrected inline and a primary error detection notification output signal is raised.
- A double-bit error raises only a primary error detection notification output signal.

An external read access encodes all signals of the bus, such as payload and control, including the AXI valid and AXI ready signals. For more information on ECC for the ACP, see [ACP bridge on page 9-40](#).

#### ———— Note ————

- ECC is only present on data buses. All other signals are protected by parity.
- ECC on the ACP bus is supported only when the ACP bridge is implemented.
- For build options where the ECC on the external ACP bus is supported, when byte line strobes are sparse on an ACP write access, the unused bytes are masked in the processor and assumed to be driven LOW. This is because the ECC is computed on a 64-bit chunk. Therefore, a master driving the ACP must compute the ECC bits together with the write data with the same assumption.

The AXI slave port is not protected by ECC. It reports any ECC error as a slave error on the AXI bus.

If ECC errors are found on TCM RAMs:

- For reads, the slave error is reported for both correctable and fatal error. If the error is fatal, the **FATALRAMERR** output is also raised.
- For writes, no response is sent. If a fatal error occurs, the **FATALRAMERR** output is raised.

You can configure whether this logic is present in the Cortex-R7 MPCore processor.

---

**Note**

To enable ECC in the Cortex-R7 MPCore processor, you must first enable ECC on the RAMs.

---

## 7.2.5 ECC codes

Table 7-1 on page 7-5 shows that three different ECC codes are required:

- A 34-bit ECC with seven check bits for the tag RAM of the data side.
- A 32-bit ECC with seven check bits for the data RAM of the data side, the tag RAM of the instruction side, and the tag RAMs of the SCU.
- A 64-bit ECC with eight check bits for the instruction data RAM.

---

**Note**

The 32-bit ECC check matrix can be a subset of the 34-bit ECC check matrix, so that only two different ECC codes are required, a 34-bit ECC and a 64-bit ECC.

---

## 7.2.6 RAM configuration

Table 7-3 shows the RAM configuration with or without ECC.

**Table 7-3 RAM configuration with or without ECC**

RAM	Storage for a RAM set without ECC	Storage for a RAM set with ECC
Data tag RAM	4x29 bits	4x(25+7) bits
Data data RAM	8x32 bits	8x(32+7) bits
Instruction tag RAM	4x21 bits	4x(21+7) bits
Instruction data RAM	4x64 bits	4x(64+8) bits
SCU tag RAMs	4x23 bits	4x(23+7) bits
Data TCM	2x32 bits	2x(32+7) bits
Instruction TCM	4x64 bits	4x(64+8) bits

Table 7-4 shows the RAM configuration with or without parity.

**Table 7-4 RAM configuration with or without parity**

RAM	Storage for a set without parity	Storage for a set with parity
BTAC	2x32+2x28 bits	2x(32+4) + 2x(28+4) bits
PRED	4x4 bits	4x(4+4) bits

### 7.2.7 Performance impact

In an error-free system, the major performance impact is the cost of the read-modify-write scheme for non-full stores in the data side. If the store buffer does not have a complete ECC chunk, it must read the word to be able to compute the check bits. The data can then be written in the RAM. This additional read can have a negative impact in performance because it prevents the slot from being used for another write.

The out-of-order and outstanding capabilities of the L1 memory system mask part of the additional read, and it is negligible for most codes. However, ARM recommends that you use as few cacheable STRB/STRH instructions as possible to reduce the performance impact.

———— **Note** ————

There might be a frequency impact because XOR trees are added on the data returned from the RAMs.

—————



## 7.3 Logic protection

The logic is protected by a duplicate processor that is the exact copy of the first processor. Both processors share the same RAMs protected with ECC and the same input pins. The second processor is delayed by two clock cycles so that this redundant system can detect glitches in the inputs.

The outputs of the two processors are compared on each cycle to detect any error. The outputs of the first processor are delayed so they can be synchronized with the second processor. This mechanism relies on the fact that any error occurring in the processor is eventually visible on the outputs of the processor, or is inherently a low-risk failure.

On detection of an error in one processor, both processors are reset before executing a code sequence, to put them in the same initial state. They can then restart execution from a previously taken snapshot.

The processor provides a template of the logic required for the comparison of the two processors.

See [Lock-step on page 7-14](#) and [Static split/lock on page 7-16](#).

## 7.4 External memory and bus protection

On the external AXI buses, control bits are protected by parity, and data is protected by ECC bits. This protection is available on the following bus interfaces:

- The AXI master ports.
- The peripheral port.
- The optional ACP slave port.

When a parity error is detected on any control bits of the AXI transfer, no correction is done, but an event is reported to the external system.

When a correctable ECC error is detected on the data bits, the data is corrected. If the ECC error is not correctable, an error event is reported.

### 7.4.1 Reporting errors

The ECC logic protecting the AXI buses assumes that an entity in the system other than the processor is responsible for reacting to any ECC error detected on the bus, to restart the system correctly or to take any related actions. For this reason, the following events are output for every protected AXI interface:

**AXICORRERR** A correctable ECC error has been detected on read or write data.

**AXIFATALERR** A fatal ECC error has been detected on one of the channels.

It is also assumed that any signal of the bus always has a defined value after reset.

## 7.5 Programmers view

The following sections describe the programming aspects of ECC:

- [Registers](#).
- [Error detection notification signals on page 7-13](#).

### 7.5.1 Registers

Various processor and SCU registers are used to enable and monitor ECC:

- [Processor registers](#).
- [SCU registers](#).

#### Processor registers

The following processor registers are used in ECC:

##### Auxiliary Control Register (ACTLR)

Bits[10:9] of this register are used to enable ECC checking. See [Auxiliary Control Register on page 4-25](#).

##### ECC Error Registers (DEER0-2/IEER0-2 and DTCMEER/ITCMEER)

These registers provide information on ECC errors. See [ECC Error Registers on page 4-39](#).

##### Performance counters

The performance counters can be configured to monitor several ECC-related metrics. See [Performance monitoring events on page 10-7](#) for more information.

##### Cache and TCM Debug Operation Register (CTDOR)

The processor contains registers that provide direct access to the caches. These registers enable the RAM analysis on error and the auto-checking of the ECC mechanisms by software. On the instruction side, these registers enable direct access to the instruction cache and to the instruction data. BTAC and PRED cannot be accessed in this way. On the data side, the tag RAM and the data cache RAM can be accessed in this way. See [Cache and TCM Debug Operation Register on page 4-36](#).

#### SCU registers

The SCU is seen as a peripheral by the processors in the Cortex-R7 MPCore processor, and has its own memory-mapped register file. The following SCU registers are used in ECC:

##### SCU Control Register

Bits[15:12] of this register are used to enable ECC checking on the AXI ports. See [SCU Control Register on page 9-6](#).

##### SCU Error Bank Registers

Bits[13:5] of these registers hold the SCU tag RAM index, and bits[1:0] show the error status. See [SCU Error Bank First Entry Register on page 9-14](#) and [SCU Error Bank Second Entry Register on page 9-15](#).

### Performance counters

Events related to the SCU are reported to the PMU of each processor. The performance counters can be configured to monitor several ECC-related metrics. See [Performance monitoring events on page 10-7](#) for more information on SCU-related events.

### SCU Debug Cache Registers

These registers provide information on various aspects of ECC for the SCU:

- The SCU Debug Tag RAM Operation Register shows the the address and action for the SCU tag RAM access.
- The SCU Debug Tag RAM Data Value Register and SCU Debug Tag RAM ECC Chunk Register contain the data from the memory selected by the SCU Debug Tag RAM Operation Register.

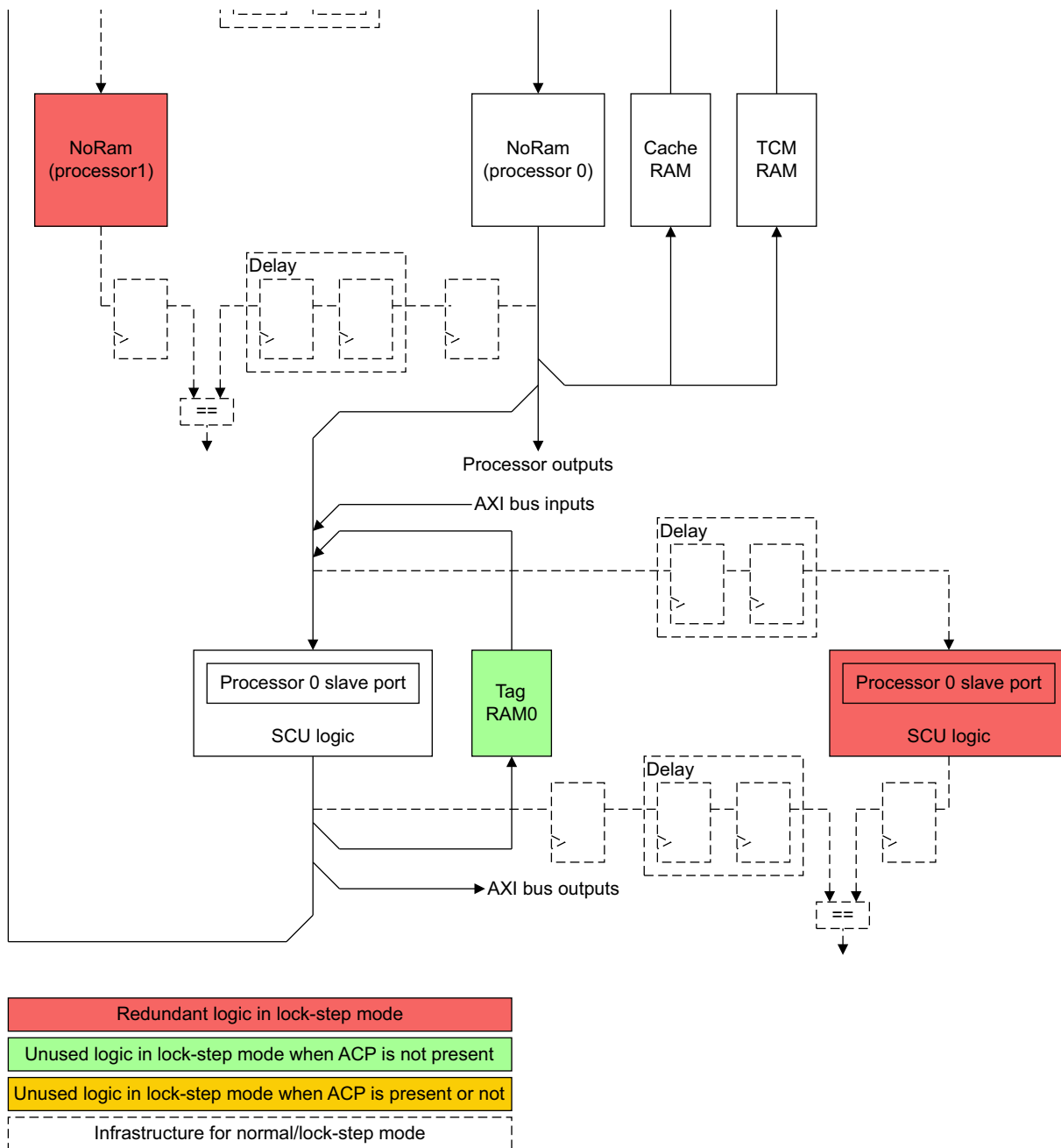
See [SCU Debug tag RAM access on page 9-16](#).

## 7.5.2 Error detection notification signals

When an error is detected, error signals are asserted to notify the external system responsible for analyzing the fault. [Error detection notification signals on page A-25](#) describes the outputs used for these events.

## 7.6 Lock-step

You can implement the Cortex-R7 MPCore processor with a second, redundant copy of the `cpu_noram`, `scu_noram`, and `axis` modules. This is known as lock-step and provides redundancy in the logic without duplicating the RAMs that are protected by ECC. Figure 7-1 shows how lock-step is implemented.



**Figure 7-1 Lock-step**

Because the dual-redundant logic has a significant impact on the area, not duplicating the RAMs minimizes this impact. Both copies of the logic run in parallel, although offset in time, and the outputs are compared to detect errors. There are two sets of comparators:

- One for `cpu_noram` output comparison.
- One for `no_cpu`, that is, `scu_noram` and `axis` output comparison.

All outputs of the noram modules are compared, except for the debug, MRP, ETM, and MBIST signals.

———— **Note** ————

**COMPENABLE** and **COMPFAULT** are global for both the processor and SCU comparators.

---

## 7.7 Static split/lock

Static split/lock enables you to choose between:

- Performance mode, that is, a multiprocessing configuration with two processors.
- Lock-step mode, that is, a lock-step configuration with dual-redundant logic.

Because this is a static split/lock, you can only switch modes during reset. The global input **SAFEMODE** enables you to choose the mode:

- **SAFEMODE** HIGH for lock-step mode.
- **SAFEMODE** LOW for performance mode.

The **SAFEMODE** input must be kept stable during normal operation and can only be changed at reset.

[Figure 7-2 on page 7-17](#) shows how split/lock is implemented.

In static split/lock, processor 1 is present with both noram and ram modules. When the processor is operating in lock-step mode, the ram logic is clamped.

The redundant no\_cpu module, that is, SCU and AXIS, is always present but it is only used in lock-step mode. It is clamped in performance mode.

---

**Note**

Only the processor 0 slave port of the SCU is duplicated. The interface with processor 1 is not required because, when in lock-step mode, the processor 1 noram is used as a redundancy of processor 0.

---

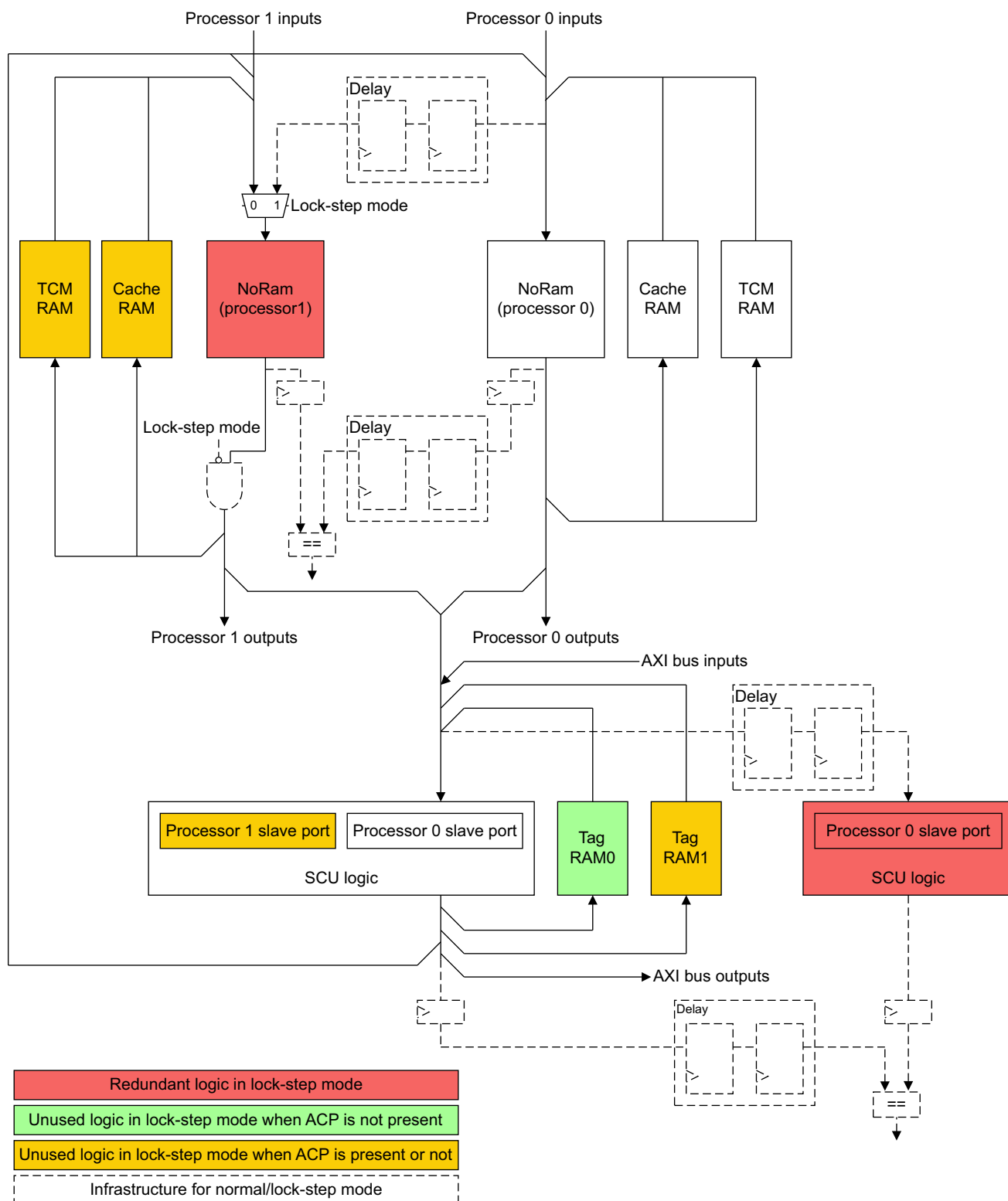


Figure 7-2 Static split/lock



# Chapter 8

## Determinism Support

This chapter describes the determinism support features of the Cortex-R7 MPCore processor. It contains the following sections:

- *About determinism support* on page 8-2.
- *Memory Protection Unit* on page 8-3.
- *Branch prediction* on page 8-11.
- *Low latency interrupt mode* on page 8-12.
- *System configurability and QoS* on page 8-13.
- *Instruction and data TCM* on page 8-15.

## 8.1 About determinism support

The Cortex-R7 MPCore processor contains a number of features that provide deterministic timing and low interrupt latency for hard real-time applications. These features are in individual processors and the SCU.

## 8.2 Memory Protection Unit

The MPU works with the L1 memory system to control accesses to and from L1 and external memory. For a full architectural description of the MPU and the memory map, see the *ARM® Architecture Reference Manual ARMv7-A and ARMv7-R edition*.

The MPU enables you to partition memory into regions and set individual protection attributes for each region. The MPU supports 12 or 16 memory regions, depending on the configuration. See the *ARM® Cortex®-R7 MPCore Configuration and Sign-off Guide*.

Each region is programmed with a base address and size, and the regions can be overlapped to enable efficient programming of the memory map. To support overlapping, the regions are assigned priorities, with region 0 having the lowest priority and region 11 or 15 having the highest priority respectively when 12 or 16 memory regions are supported. The MPU returns access permissions and attributes for the highest priority region where the address hits.

The MPU is programmed using CP15 registers c1 and c6, see [MPU memory region programming registers on page 4-28](#). Memory region control read and write access is permitted only from privileged modes.

This section describes:

- [Regions](#).
- [Memory types on page 8-6](#).
- [Region attributes on page 8-7](#).
- [MPU interaction with memory system on page 8-9](#).
- [MPU faults on page 8-10](#).
- [MPU software-accessible registers on page 8-10](#).

### 8.2.1 Regions

The MPU regions are:

- [Memory regions](#).
- [Overlapping regions on page 8-4](#).
- [Background regions on page 8-6](#).
- [TCM regions on page 8-6](#).

#### Memory regions

For more information on how to enable or disable the MPU, see [MPU interaction with memory system on page 8-9](#).

Depending on the implementation, the MPU has a maximum of 12 or 16 regions. You can specify the following for each region:

- [Region base address](#).
- [Region size on page 8-4](#).
- [Subregions on page 8-4](#).
- [Region attributes on page 8-4](#).
- [Region access permissions on page 8-4](#).

#### Region base address

The base address defines the start of the memory region. You must align this to a region-sized boundary. For example, if a region size of 8KB is programmed for a given region, the base address must be a multiple of 8KB.

---

**Note**

---

If the region is not aligned correctly, this results in UNPREDICTABLE behavior.

---

**Region size**

The region size is specified as a 5-bit value, encoding a range of values from 256 bytes to 4GB. [Table 4-29 on page 4-30](#) shows the encoding.

**Subregions**

Each region can be split into eight equal sized non-overlapping subregions. An access to a memory address in a disabled subregion does not use the attributes and permissions defined for that region. Instead, it uses the attributes and permissions of a lower priority region or generates a background fault if no other regions overlap at that address. This enables increased protection and memory attribute granularity.

**Region attributes**

Each region has a number of attributes associated with it. See [Memory types on page 8-6](#) for more information about memory types, and [Region attributes on page 8-7](#) for a description of how to assign types and attributes to a region.

**Region access permissions**

Each region can be given no access, read-only access, or read/write access permissions for privileged or all modes. In addition, each region can be marked as *eXecute Never* (XN) to prevent instructions being fetched from that region.

For example, if a user mode application attempts to access a *Privileged mode access only* region, a permission fault occurs.

The ARM architecture uses constants known as *inline literals* to perform address calculations. The assembler and compiler automatically generate these constants and they are stored inline with the instruction code. To ensure correct operation, only a memory region that has permission for data read access can execute instructions. For more information, see the *ARM® Architecture Reference Manual ARMv7-A and ARMv7-R edition*. For information about how to program access permissions, see [Table 4-31 on page 4-32](#).

**Overlapping regions**

You can program the MPU with two or more overlapping regions. For overlapping regions, a fixed priority scheme determines attributes and permissions for memory access to the overlapping region. Attributes and permissions for region 11 take highest priority, those for region 0 take lowest priority. For example:

<b>Region 2</b>	Is 4KB in size, starting from address 0x3000. Privileged mode has full access, and user mode has read-only access.
<b>Region 1</b>	Is 16KB in size, starting from address 0x0000. Both privileged and user modes have full access.

When the processor performs a data write to address 0x3010 while in user mode, the address falls into both region 1 and region 2, as [Figure 8-1 on page 8-5](#) shows. Because these regions have different permissions, the permissions associated with region 2 are applied. Because user mode is read access only for this region, a permission fault occurs, causing a data abort.

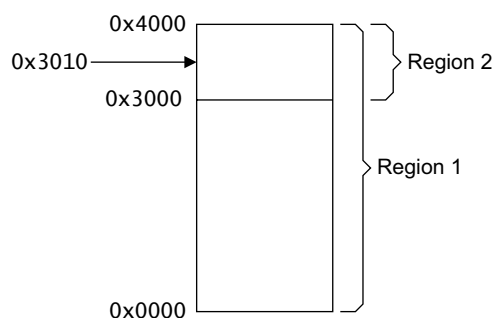


Figure 8-1 Overlapping memory regions

**Example of using regions that overlap**

You can use overlapping regions for stack protection, as [Figure 8-2](#) shows. For example:

- Allocate to region 1 the appropriate size for all stacks.
- Allocate to region 2 the minimum region size, 256 bytes, and position it at the end of the stack for the current process.
- Set the region 2 access permissions to No Access.

If the current process overflows the stack it uses, a write access to region 2 by the processor causes the MPU to raise a permission fault.

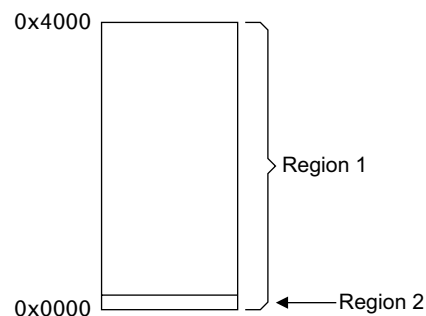


Figure 8-2 Overlay for stack protection

**Example of using subregions**

You can use subregions for stack protection, as [Figure 8-3 on page 8-6](#) shows. For example:

- Allocate to region 1 the appropriate size for all stacks.
- Set the least-significant subregion disable bit. That is, set the subregion disable field, bits[15:8], of the CP15 MPU Region Size Register to 0x01.

If the current process overflows the stack it uses, a write access by the processor to the disabled subregion causes the MPU to raise a background fault.

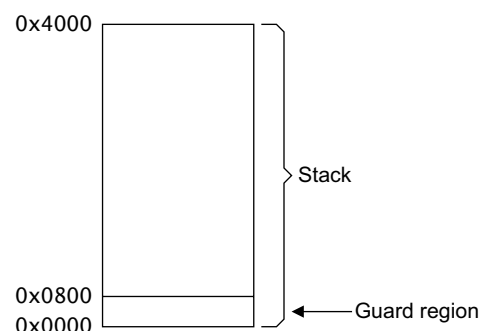


Figure 8-3 Overlapping subregion of memory

### Background regions

Overlapping regions increase the flexibility of how the regions can be mapped onto physical memory devices in the system. You can also use the overlapping properties to specify a background region. For example, you might have a number of physical memory areas sparsely distributed across the 4GB address space. If a programming error occurs, the processor might issue an address that does not fall into any defined region.

If the address that the processor issues falls outside any of the defined regions, the MPU is hard-wired to abort the access. That is, all accesses for an address that is not mapped to a region in the MPU generate a background fault. You can override this behavior by programming region 0 as a 4GB background region. In this way, if the address does not fall into any of the other 11 or 15 regions, the attributes and access permissions you specified for region 0 control the access.

In privileged modes, you can also override this behavior by setting the BR bit, bit[17], of the SCTLRR. This causes privileged accesses that fall outside any of the defined regions to use the default memory map. User mode accesses to this background region cause faults.

### TCM regions

Any memory address that you configure to be accessed using a TCM is mapped as having Normal, Non-shareable type attributes, regardless of the attributes of any MPU region that the address also belongs to. Access permissions for an address in a TCM region are preserved from the MPU region that the address also belongs to. For more information, see [System configurability and QoS on page 8-13](#) and [Instruction and data TCM on page 8-15](#).

## 8.2.2 Memory types

The ARM Architecture defines a set of memory types with characteristics that are suited to particular devices. There are three mutually exclusive memory type attributes:

- Strongly Ordered.
- Device.
- Normal.

MPU memory regions must each be assigned a memory type attribute. [Table 8-1](#) shows a summary of the memory types.

**Table 8-1 Memory attributes summary**

Memory type attribute	Shareable or Non-shareable	Other attributes	Description
Strongly Ordered	-	-	All memory accesses to Strongly Ordered memory occur in program order. All Strongly Ordered accesses are assumed to be shareable.
Device	Shareable	-	For memory-mapped peripherals that two processors share.
	Non-shareable	-	For memory-mapped peripherals that only a single processor uses.
Normal	Shareable	Non-cacheable Write-back Cacheable	For normal memory that is shareable between two processors.
	Non-shareable	Non-cacheable Write-back Cacheable	For normal memory that only a single processor uses.

For more information on memory attributes and types, memory barriers, and ordering requirements for memory accesses, see the *ARM® Architecture Reference Manual ARMv7-A and ARMv7-R edition*.

### Using memory types

The memory system contains a store buffer. This helps to improve the throughput of accesses to Normal type memory. Because of the ordering rules that they must follow, accesses to other types of memory typically have a lower throughput and higher latency than accesses to Normal memory. In particular, reads from Device or Strongly Ordered memory must first drain the store buffer of all writes to Device or Strongly Ordered memory:

Similarly, when it is accessing Strongly Ordered or Device type memory, the processor's response to interrupts is modified, and the interrupt response latency is longer.

To ensure optimum performance, you must understand the architectural semantics of the different memory types. Use Device memory type for appropriate memory regions, typically peripherals, and only use Strongly Ordered memory type for memory regions where it is essential.

### 8.2.3 Region attributes

Each region has a number of attributes associated with it. These control how a memory access is performed when the processor accesses an address that falls within a given region. The attributes are:

- Memory type, see [Memory types on page 8-6](#), one of:
  - Strongly Ordered.
  - Device.
  - Normal.
- Shareable or Non-shareable.
- Non-cacheable.
- Write-back write allocate.

The Region Access Control Registers use five bits to encode the memory region type. These are the TEX[2:0], C, and B bits. [Table 8-2](#) shows the mapping of these bits to memory region attributes.

———— **Note** ————

In earlier versions of the architecture, the TEX, C, and B bits were known as the Type Extension, Cacheable and Bufferable bits. These names no longer adequately describe the function of the B, C, and TEX bits.

In addition, the [MPU Region Access Control Registers on page 4-31](#) contain the shareable bit, S. This bit usually determines whether the memory region is Shareable (1) or Non-shareable (0). However, in some cases, the shareable attribute is forced by other attributes, for example, Strongly Ordered memory types are always Shareable.

**Table 8-2 TEX[2:0], C, and B encodings**

TEX[2:0]	C	B	Description	Memory type	Shareable?
000	0	0	Strongly Ordered.	Strongly Ordered	Shareable
000	0	1	Shareable Device.	Device	Shareable
000	1	0	Reserved.	-	-
001	0	0	Outer and Inner Non-cacheable.	Normal	S bit <sup>a</sup>
001	0	1	Reserved.	-	-
001	1	0			
001	1	1	Outer and Inner write-back, write-allocate.	Normal	S bit <sup>a</sup>
010	0	0	Non-shareable Device.	Device	Non-shareable
010	0	1	Reserved.	-	-
010	1	X			
011	X	X			
1BB	A	A	Cacheable memory: AA <sup>b</sup> Inner policy. BB <sup>b</sup> Outer policy.	Normal	S bit <sup>a</sup>

a. Region is Shareable if S == 1, and Non-shareable if S == 0.

b. [Table 8-3 on page 8-9](#) shows the encoding for these bits.

### Cacheable memory policies

When TEX[2] == 1, the memory region is cacheable memory, and the rest of the encoding defines the Inner and Outer cache policies:

**TEX[1:0]** Defines the Outer cache policy.

**C,B** Defines the Inner cache policy.



The same encoding is used for the Outer and Inner cache policies. Table 8-3 shows the encoding.

**Table 8-3 Inner and Outer cache policy encoding**

Memory attribute encoding	Cache policy
00	Non-cacheable
01	Write-back, write-allocate
10	Reserved

When the processor performs a memory access through its AXI3 bus master interface:

- The Inner attributes are indicated on the **A\*USERM** signals.
- The Outer attributes are indicated on the **A\*CACHEM** signals.

For more information on region attributes, see the *ARM® Architecture Reference Manual ARMv7-A and ARMv7-R edition*.

## 8.2.4 MPU interaction with memory system

This section describes how to enable and disable the MPU. After you enable or disable the MPU, the pipeline must be flushed using ISB and DSB instructions to ensure that all subsequent instruction fetches see the effect of turning on or off the MPU.

Before you enable or disable the MPU you must:

1. Program all relevant CP15 registers. This includes setting up at least one memory region that covers the currently executing code, and that provides read and execute permissions in at least privileged mode.
2. Invalidate the instruction cache.
3. Enable the instruction cache.
4. Invalidate the data cache.

The following code is an example of enabling the MPU:

```
MRC p15, 0, R1, c1, c0, 0    ; read CP15 register 1
ORR R1, R1, #0x1
DSB
MCR p15, 0, R1, c1, c0, 0    ; enable MPU
ISB
Fetch from programmed memory map
Fetch from programmed memory map
Fetch from programmed memory map
Fetch from programmed memory map
```

The following code is an example of disabling the MPU:

```
MRC p15, 0, R1, c1, c0, 0    ; read CP15 register 1
BIC R1, R1, #0x1
DSB
MCR p15, 0, R1, c1, c0, 0    ; disable MPU
ISB
Fetch from default memory map
Fetch from default memory map
Fetch from default memory map
Fetch from default memory map
```

The MPU does not check accesses from the AXI TCM slave. You can configure the processor to enable access to the TCM interfaces from the AXI TCM slave port.

For more information on the default memory map, see the *ARM® Architecture Reference Manual ARMv7-A and ARMv7-R edition*.

### 8.2.5 MPU faults

The MPU can generate these types of fault:

- [Background fault](#).
- [Permission fault](#).
- [Alignment fault](#).

When a fault occurs, the memory access or instruction fetch is synchronously aborted, and a prefetch abort or data abort exception is taken as appropriate. No memory accesses are performed on the AXI3 bus master interface. See [Fault handling on page 6-3](#) for more information on fault handling.

#### Background fault

A background fault is generated when the MPU is enabled and a memory access is made to an address that is not within an enabled subregion of an MPU region. A background fault does not occur if the background region is enabled and the access is Privileged.

#### Permission fault

A permission fault is generated when a memory access does not meet the requirements of the permissions defined for the memory region that it accesses. See [Region access permissions on page 8-4](#).

#### Alignment fault

An alignment fault is generated if a data access is performed to an address that is not aligned for the size of the access, and strict alignment is required for the access. A number of instructions that access memory, for example, LDM and STC, require strict alignment. See the *ARM® Architecture Reference Manual ARMv7-A and ARMv7-R edition* for more information. In addition, strict alignment can be required for all data accesses by setting the A-bit in the System Control Register. See [System Control Register on page 4-23](#).

### 8.2.6 MPU software-accessible registers

[MPU memory region programming registers on page 4-28](#) shows the CP15 registers that control the MPU.

## 8.3 Branch prediction

The branch prediction uses both static and dynamic techniques to predict:

- That there is a branch instruction at a given address.
- The type of the branch:
  - Unconditional or conditional.
  - Immediate or load.
  - Normal branch, function call, or function return.
- The target address or the state of the branch, either ARM or Thumb.
- The direction of conditional branch, either taken or not taken.

The static branch prediction is based on decoding the instruction. Therefore, it can see branches on fresh code, without any history, but the prediction is done only at the decoding stage, so no fetch decision can be made before this stage, that is, speculative fetches cannot be made.

The dynamic branch prediction estimates the instructions based on history, so that it can fetch speculatively to an arbitrary chosen branch of the execution code. More hardware is required, but it saves some unnecessary i-cache look-up/memory accesses, and the prediction quality is higher for previously seen branches.

By default, the dynamic branch prediction is used and, if there is no information in its history, the static prediction is used instead.

## 8.4 Low latency interrupt mode

The low latency interrupt mode can be enabled or disabled. See the [System Control Register on page 4-23](#) for information about how to program the interrupt latency mode. The fast interrupt bit controlling the interrupt mode is disabled by default to allow some enhanced performance, and can be modified if you require a higher level of control on the determinism. By enabling fast low latency interrupt mode, entry into an interrupt routine is slightly quicker, but with a slight reduction in global processor performance.

When the low latency interrupt mode is disabled, the interrupts are inserted in the decoder stage and are seen as a branch instruction targeting the interrupt vector. This means that all instructions in the pipeline must finish their execution before starting to execute new instructions from the interrupt handler. When those instructions in the pipeline depend on a load that misses, this time depends on the external memory latency. Another case of instructions that might take time to complete are long instructions such as VDIV and VSQRT. This mode enables better speculative instruction execution, and therefore better average performance.

When the low latency interrupt mode is enabled, the following are flushed:

- All loads and stores that have not started.
- Those loads and stores that have started to normal memory, and are still speculative.
- FDIV and VSQRT operations.
- Any pending CP15 operations with CRn=7.
- Any pending DMB or DSB operations.
- Instructions that follow these that are already in the pipeline.

### ———— Note ————

The following are not flushed when they have started:

- Loads/stores to Strongly Ordered or Device memory region.
- Swap accesses. These accesses are deprecated in the ARMv7 architecture.

This behavior has the following effect on the data side:

- The ability to flush instructions requires keeping them speculative for their lifetime, because the register renaming of the integer core requires a recovery mechanism. This impacts the average performance by 3-4%.
- This frees up resources in the pipeline, and in the four slots of the LSU, accelerating the handling of the interrupt routine.

The processor LSU supports up to four accesses so that, for example, a load with a significant memory latency does not block a subsequent load/store access requested by the integer core. This is the normal behavior when the low latency interrupt mode is disabled. When the low latency interrupt mode is enabled, [Table 8-4](#) shows that Strongly Ordered and Device read accesses, in addition to all store accesses, have an effect on the performance because they wait for cacheable loads to have their data returned.

**Table 8-4 Performance and determinism effects in low latency interrupt mode**

Low latency interrupt mode	IPC performance	Level of determinism
Disabled	High	Medium
Enabled	High minus 3-4%	High

## 8.5 System configurability and QoS

You can use the *Quality of Service* (QoS) to ensure that low priority cacheable traffic does not block the flow of accesses from peripherals and AXI master port 1.

### Caution

If the processor uses the QoS feature and address filtering is enabled for AXI master port 1, the slave connected to AXI master port 1 must be private to the processor. When QoS is not enabled, no such system constraint exists.

A real-time system with two AXI3 master ports and address filtering can stream critical tasks and background tasks so that the flow of background tasks, particularly cached low priority tasks, that can have significant memory latency, does not block the flow of critical tasks:

- High priority traffic consists of transfers accessing either AXI master port 1 when used with address filtering, the peripheral port, or the TCM
- Any other transfers going through AXI master port 0, are considered to have low priority.

The QoS bit in the *Auxiliary Control Register* on page 4-25 is used to enable QoS:

- If this bit is set, some hardware resources are blocked for low priority traffic in the processor. This means that the high priority traffic has the necessary resources to start its execution, typically after an interrupt has occurred. As soon as the low priority traffic has completed its pending transfers, the high priority traffic can use all the hardware resources.
- If this bit is not set, no hardware resources are blocked for low priority traffic, and both the low and high priority traffics share and use all the available resources. This configuration has better average performance, because all hardware resources are available to all traffic.

The QoS bit can be used to ensure that low priority cacheable traffic does not block the flow of accesses from the following:

- Peripherals connected on the peripheral port.
- Data TCM accesses.
- Cacheable traffic that is connected on the optional external AXI master port 1 when used with address filtering.

You can set the QoS bit on a per processor basis to ensure that low priority cacheable traffic with significant memory latencies does not block the flow of traffic from these tasks. The SCU offers some QoS as soon as the filtering is enabled on AXI master port 1.

You can use the QoS bit to set different mixes of traffic flows:

- If the QoS bit is not set, all traffic can use all hardware resources regardless of priority.
- If the QoS bit is set, low priority traffic cannot use all the hardware resources.

Table 8-5 shows the recommended QoS bit settings according to traffic types.

**Table 8-5 Recommended QoS bit settings according to traffic types**

Traffic flow types	Low priority cacheable traffic types	
	Small and bounded memory latency	Potentially large and unbounded memory latency
Peripheral and TCM traffic only.	Do not set QoS bit	Do not set QoS bit
Peripheral and low priority cacheable traffic, the TCM can be present or not, and low priority traffic is on AXI master port 0.	Do not set QoS bit	Set QoS bit
Peripheral traffic, low and high priority cacheable traffic, the TCM can be present or not, and low priority traffic is on AXI master port 0, with high priority traffic on AXI master port 1.	Do not set QoS bit	Set QoS bit

The recommendations in Table 8-5 make the following assumptions:

- The design implements AXI master port 0 and AXI master port 1 with address filtering.
- High priority traffic consists of transfers accessing either the local SRAM on AXI master port 1, the peripheral port, or the TCM.
- Other transfers, namely the cacheable transfers going through the AXI master port 0, have low priority.

## 8.6 Instruction and data TCM

Instruction and data TCMs are tightly-coupled in the Cortex-R7 MPCore processor. There are no external ports for the TCMs and only SRAM memory is supported. Instructions cannot be stored in the Data TCM. An instruction fetch to the Data TCM goes to the AXI interface and not the Data TCM. ARM recommends that the DTCM memory region is marked as XN in the MPU region settings to prevent instruction accesses to this address range.

There is an option to permit a single wait state on the instruction TCM. The data TCM does not accommodate wait states.

You can configure the instruction and data TCM size and the optional instruction TCM wait state during integration. See the *ARM® Cortex®-R7 MPCore Integration Manual* for more information. The permissible TCM sizes are:

- 0KB.
- 4KB.
- 8KB.
- 16KB.
- 32KB.
- 64KB.
- 128KB.

Both TCMs can be preloaded using the AXI slave port. This slave port provides access to the TCMs only. See [AXI TCM slave port on page 2-21](#).

From a programmer's view:

- MPU regions targeting the TCM are private to the processor and non-shareable regions from a multiprocessing aspect. They are not part of the L1 data cache coherent domain.
- The size of the TCM interfaces is visible to software in the TCM Region Registers. See [DTCM Region Register on page 4-33](#) and [ITCM Region Register on page 4-34](#). If the TCM size does not match a power of 2 value, the TCM size must be the next power of 2 value above the physical memory size. If some accesses in the MPU region are not physically connected to the TCM, you can choose how to drive read data for the address range uncovered by the physical TCM, for example, alias or drive as 0.

Both instruction and data TCM are ECC protected. For more information, see [ECC on RAMs on page 7-5](#).

---

### Note

---

Write accesses to the instruction TCM are possible for debug purposes, but with limited throughput.

---

# Chapter 9

## Multiprocessing

This chapter describes the multiprocessing features of the Cortex-R7 MPCore processor, including the SCU. It contains the following sections:

- *About multiprocessing and the SCU on page 9-2.*
- *Multiprocessing programmers view on page 9-4.*
- *SCU registers on page 9-5.*
- *Interrupt controller on page 9-19.*
- *Private timer and watchdog on page 9-29.*
- *Global timer on page 9-35.*
- *Accelerator Coherency Port on page 9-39.*



## 9.1 About multiprocessing and the SCU

The SCU connects Cortex-R7 processors to the memory system through the AXI3 interfaces. The SCU functions are to:

- Maintain data cache coherency between the Cortex-R7 processors.
- Initiate L2 AXI3 memory accesses.
- Arbitrate between Cortex-R7 processors requesting L2 accesses.
- Manage ACP accesses, with data cache coherency for the processors.

---

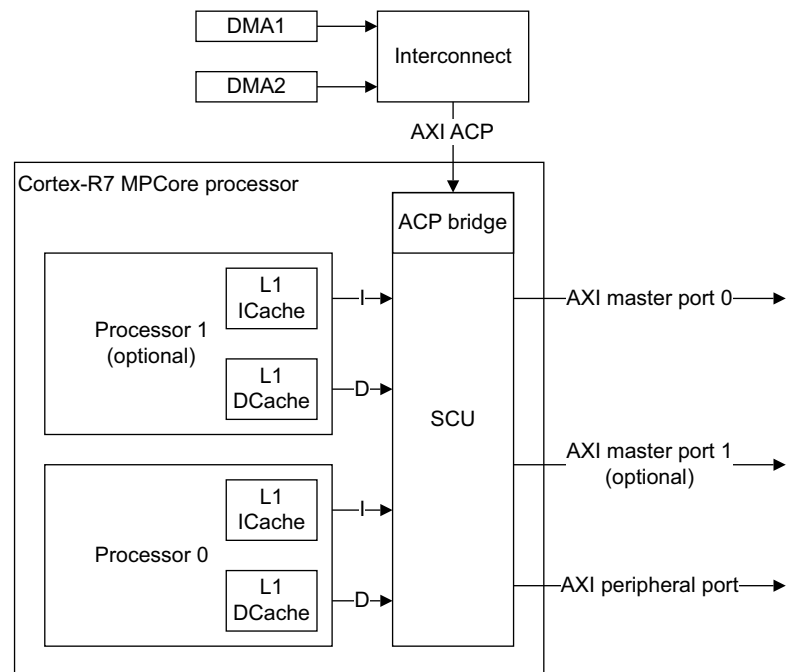
### Note

---

The Cortex-R7 SCU does not support hardware management of coherency of the instruction cache.

---

The SCU has an optional *Accelerator Coherency Port (ACP)* used to connect a non-cached master such as a DMA to the Cortex-R7 MPCore processor, as shown in [Figure 9-1](#).



**Figure 9-1 SCU and ACP**

The SCU has two slave ports per processor, one connected to the instruction bus and one connected to the data bus of each processor.

The data transfers on the ACP can be coherent in a multiprocessor implementation. The ACP also provides a noncoherent mode, where any transfer can be done directly with the level 2 memory directly, without having to deal with data coherency. See [Coherent and noncoherent mode on page 9-39](#) for information on coherent and noncoherent mode usage.

The data coherency efficiency, either using the ACP or between individual processors, is enhanced by using the replicated Tag RAMs of each L1 Data Cache Tag RAM of the individual processors in the multiprocessor implementation. Any transfer through the SCU is looked up in the replicated Tag RAM to determine whether data is present in the L1 data cache of either processor without having to access that cache. The replicated Tag RAM is an implementation of a directory structure.

You can choose to have one or two AXI master ports. The SCU also provides memory-mapped address filtering to enable you to route specific transfers on AXI master port 1 with QoS guarantees. See [AXI master port 1](#) on page 2-19 and [System configurability and QoS](#) on page 8-13.

The SCU also contains:

- A memory-mapped peripheral port, see [AXI peripheral port](#) on page 2-20
- Memory-mapped control registers, see [SCU registers](#) on page 9-5.
- An interrupt controller, see [Interrupt controller](#) on page 9-19.
- Private timers and watchdog, see [Private timer and watchdog](#) on page 9-29.
- Global timers, see [Global timer](#) on page 9-35.

You can configure the individual processor event monitors to gather statistics on the operation of the SCU. See [Performance Monitoring Unit](#) on page 10-3.

## 9.2 Multiprocessing programmers view

To enable data cache coherency in the processors:

- Set the SMP bit, bit[6] in the [Auxiliary Control Register on page 4-25](#). The reset value is zero.
- Set the SCU enable bit, bit[0] in the [SCU Control Register on page 9-6](#). The reset value is zero.

The L1 data cache coherency between the processors is done in the inner Write-Back, Write-Allocate Shareable memory regions. To enable the coherency on the ACP, see [Coherent and noncoherent mode on page 9-39](#).

### 9.3 SCU registers

All SCU registers are memory-mapped and have a common base address. Addresses are relative to the base address of the region for the SCU memory map, **PERIPHBASE[31:13]**. To access the SCU registers, **PERIPHBASE[31:13]** must be located in the address range that **PFILTERSTART[31:20]** and **PFILTEREND[31:20]** define. The value of **PERIPHBASE[31:13]** can be retrieved by a processor using the *Configuration Base Address Register* (CBAR) so that software can determine the location of the SCU registers.

The Peripheral End Address filtering must be greater than or equal to the Peripheral Start Address. See *Peripherals Filtering Start Address Register* on page 9-12 and *Peripherals Filtering End Address Register* on page 9-13 for information about these filtering addresses. The memory space in MB used for the address filtering is defined as follows:

Memory\_space (MB) = End – Start + 1.

Table 9-1 shows the peripheral accesses relative to the Peripheral Address setting.

**Table 9-1 Peripheral accesses**

Access	SCU registers		Peripheral port traffic
	Accessible by any processor	Accessible through the ACP	Accessible by any processor or through the ACP
Peripheral End Address less than Peripheral Start Address	No	No	Not enabled
Peripheral End Address equal to Peripheral Start Address	Yes	No	Enabled
Peripheral End Address greater than Peripheral Start Address	Yes	No	Enabled

Table 9-2 shows the SCU registers. All SCU registers are byte accessible and are reset by **nSCURESET**.

**Table 9-2 SCU registers summary**

Offset from PERIPHBASE[31:13]	Name	Reset value	Page
0x00	SCU Control Register	Implementation defined	<a href="#">page 9-6</a>
0x04	SCU Configuration Register	Implementation defined	<a href="#">page 9-7</a>
0x08	SCU CPU Power Status Register	-	<a href="#">page 9-9</a>
0x0C	SCU Invalidate All Registers	0x0	<a href="#">page 9-10</a>
0x40	Master Filtering Start Address Register	Defined by <b>MFILTERSTART</b> input	<a href="#">page 9-11</a>
0x44	Master Filtering End Address Register	Defined by <b>MFILTEREND</b> input	<a href="#">page 9-11</a>
0x48	Peripherals Filtering Start Address Register	Defined by <b>PFILTERSTART</b> input	<a href="#">page 9-12</a>
0x4C	Peripherals Filtering End Address Register	Defined by <b>PFILTEREND</b> input	<a href="#">page 9-13</a>
0x50	SCU Access Control Register	0b11	<a href="#">page 9-13</a>
0x60	SCU Error Bank First Entry Register <sup>a</sup>	-	<a href="#">page 9-14</a>

Table 9-2 SCU registers summary (continued)

Offset from PERIPHBASE[31:13]	Name	Reset value	Page
0x64	SCU Error Bank Second Entry Register <sup>a</sup>	-	<a href="#">page 9-15</a>
0x70	SCU Debug Tag RAM Operation Register	-	<a href="#">page 9-16</a>
0x74	SCU Debug Tag RAM Data Value Register	-	<a href="#">page 9-17</a>
0x78	SCU Debug Tag RAM ECC Chunk Register <sup>a</sup>	-	<a href="#">page 9-18</a>

a. This register is present only when ECC is implemented.

### 9.3.1 SCU Control Register

The SCU Control Register characteristics are:

**Purpose**

Enables:

- Speculative linefills to L2 with the L2C-310 Cache Controller.
- IC standby mode.
- SCU standby mode.
- SCU tag RAM ECC support.
- Address filtering.
- Bus ECC and parity control.
- Access control on master ports.
- Cache coherency features.

**Usage constraints**

This register is writable if the relevant bit in the SAC register is set.

**Configurations**

Available in all Cortex-R7 MPCore configurations.

**Attributes**

See the register summary in [Table 9-2 on page 9-5](#).

[Figure 9-2](#) shows the SCU Control Register bit assignments.

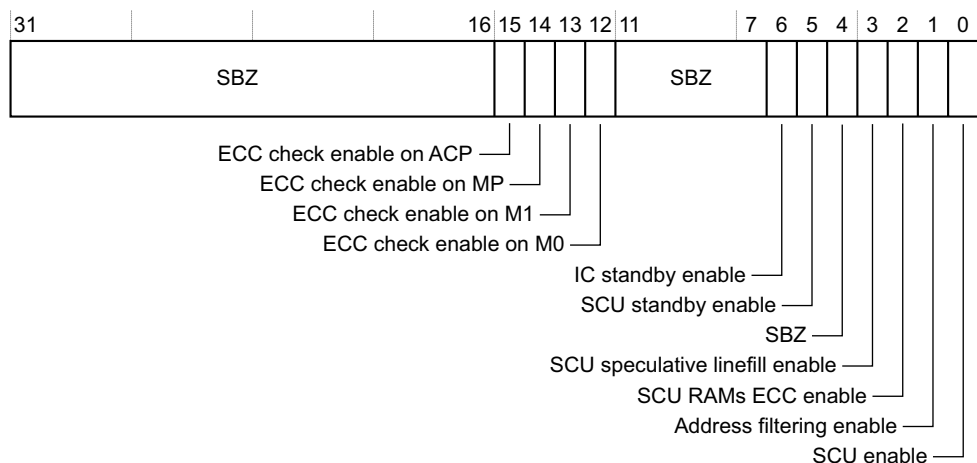


Figure 9-2 SCU Control Register bit assignments

Table 9-3 shows the SCU Control Register bit assignments.

**Table 9-3 SCU Control Register bit assignments**

Bits	Name	Description
[31:16]	Reserved	SBZ
[15]	ECC check enable on ACP	When set, enables ECC check on the <i>Accelerator Coherency Port</i> (ACP).
[14]	ECC check enable on MP	When set, enables ECC check on the AXI master peripheral port..
[13]	ECC check enable on M1	When set, enables ECC check on AXI master port 1.
[12]	ECC check enable on M0	When set, enables ECC check on AXI master port 0.
[11:7]	Reserved	SBZ
[6]	IC standby enable	When set, this stops the interrupt controller clock when no interrupts are pending, and neither processor is performing a read/write request.
[5]	SCU standby enable	When set, the clock in the SCU is turned off when all processors are in WFI mode or in powerdown, there is no pending request on the ACP, if implemented, and there is no remaining activity in the SCU.  When the clock in the SCU is off, <b>ARREADY</b> , <b>AWREADY</b> , and <b>WREADY</b> on the ACP are forced LOW. The clock is turned on when any processor leaves WFI mode, or if there is a new request on the ACP.
[4]	Reserved	SBZ
[3]	SCU speculative linefills enable	When set, coherent linefill requests are sent speculatively to the L2C-310 Cache Controller in parallel with the tag look-up.  If the tag look-up misses, the confirmed linefill is sent to the L2C-310 Cache Controller and receives RDATA earlier because the data request was already initiated by the speculative request. This feature works only if there is an L2C-310 Cache Controller in the design. When filtering is enabled only port 0 can receive speculative linefills.
[2]	SCU RAMs ECC enable	Enables ECC: 1           ECC on. 0           ECC off. This is the default setting. This bit is always zero if support for ECC is not implemented.
[1]	Address filtering enable	This is a read-only bit that indicates address filtering: 1           Address filtering on. 0           Address filtering off. This value is the value of <b>MFILTEREN</b> sampled when <b>nSCURESET</b> is deasserted. This bit is always zero if the SCU is implemented in the single master port configuration. See <a href="#">AXI master port 1 on page 2-19</a> .
[0]	SCU enable	Enables SCU: 1           SCU enabled. 0           SCU disabled. This is the default setting.

### 9.3.2 SCU Configuration Register

The SCU Configuration Register characteristics are:

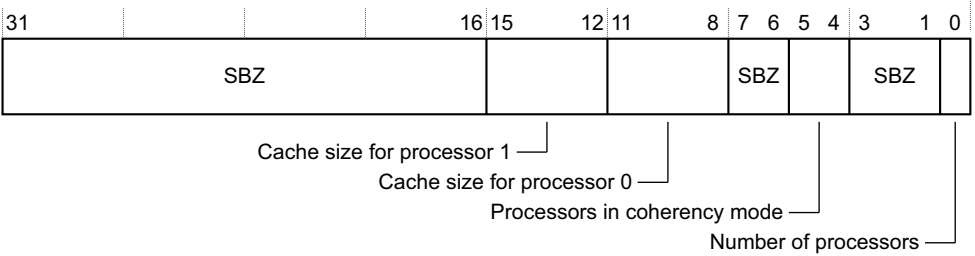
<b>Purpose</b>	<ul style="list-style-type: none"> <li>Reads tag RAM sizes for the Cortex-R7 processors that are present.</li> <li>Determines the Cortex-R7 processors that are taking part in coherency.</li> <li>Reads the number of Cortex-R7 processors present.</li> </ul>
----------------	---

**Usage constraints** This register is read-only.

**Configurations** Available in all configurations.

**Attributes** See the register summary in [Table 9-2 on page 9-5](#).

[Figure 9-3](#) shows the SCU Configuration Register bit assignments.



**Figure 9-3 SCU Configuration Register bit assignments**

[Table 9-4](#) shows the SCU Configuration Register bit assignments.

**Table 9-4 SCU Configuration Register bit assignments**

Bits	Name	Description
[31:16]	Reserved	SBZ
[15:12]	Cache size for processor 1	The encoding is as follows: 0b0101 64KB cache. 0b0100 32KB cache. 0b0011 16KB cache. 0b0010 8KB cache. 0b0001 4KB cache. 0b0000 0KB cache. All other values are Reserved.
[11:8]	Cache size for processor 0	The encoding is as follows: 0b0101 64KB cache. 0b0100 32KB cache. 0b0011 16KB cache. 0b0010 8KB cache. 0b0001 4KB cache. 0b0000 0KB cache. All other values are Reserved.
[7:6]	Reserved	SBZ

Table 9-4 SCU Configuration Register bit assignments (continued)

Bits	Name	Description
[5:4]	Processors in coherency mode	Shows the Cortex-R7 processors that are in <i>Symmetric Multi-processing</i> (SMP) or <i>Asymmetric Multi-processing</i> (AMP) mode: 1 This Cortex-R7 processor is in SMP mode taking part in coherency. 0 This Cortex-R7 processor is in AMP mode not taking part in coherency or not present. Bit 5 is for processor 1 Bit 4 is for processor 0.
[3:1]	Reserved	SBZ
[0]	Number of processors	Number of processors present in the Cortex-R7 MPCore processor: 1 Two Cortex-R7 processors, processor 0 and processor 1. 0 One Cortex-R7 processor, processor 0.

### 9.3.3 SCU CPU Power Status Register

The SCU CPU Power Status Register characteristics are:

- Purpose** Specifies the state of the Cortex-R7 processors with reference to power modes.
- Usage constraints** Writes to this register are enabled when the access bit for the processor is set in the SCU Access Control Register. See [SCU Access Control Register on page 9-13](#).  
Dormant mode and powered-off mode are controlled by an external power controller. SCU CPU Status Register bits indicate to the external power controller the power domains that can be powered down.  
Before entering any other power mode than Normal, the processor must set its status field to signal to the power controller the mode it is about to enter. The processor powerdown routine must then execute a DSB instruction and then a WFI entry instruction. When in WFI state, the **PWRCTL**On bus is enabled and signals to the power controller what it must do with power domains. See also [Individual processor power management on page 2-13](#).  
The SCU CPU Power Status Register bits can also be read by a processor exiting low-power mode to determine its state before executing its reset setup.
- Configurations** Available in all configurations.
- Attributes** See the register summary in [Table 9-2 on page 9-5](#).

Figure 9-4 shows the SCU CPU Power Status Register bit assignments.

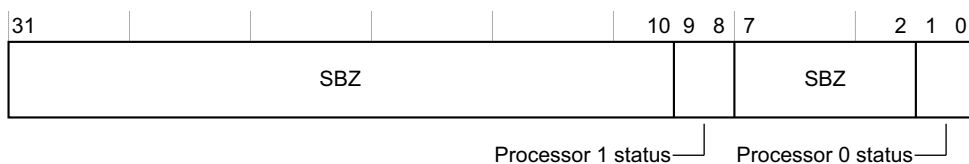


Figure 9-4 SCU CPU Power Status Register bit assignments



Table 9-5 shows the SCU CPU Power Status Register bit assignments.

**Table 9-5 SCU CPU Power Status Register bit assignments**

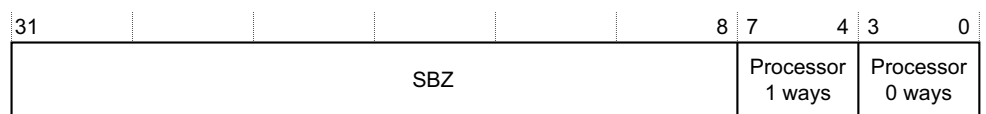
Bits	Name	Description
[31:10]	Reserved	SBZ
[9:8]	Processor 1 status	Power status of processor 1: 0x Processor must be powered on. 10 Processor can enter dormant mode. 11 Processor can enter powered-off mode.
[7:2]	Reserved	SBZ
[1:0]	Processor 0 status	Power status of processor 0: 0x Processor must be powered on. 10 Processor can enter dormant mode. 11 Processor can enter powered-off mode.

### 9.3.4 SCU Invalidate All Register

The SCU Invalidate All Register characteristics are:

<b>Purpose</b>	Invalidates the SCU tag RAMs on a per Cortex-R7 processor and per way basis.
<b>Usage constraints</b>	<ul style="list-style-type: none"> <li>This register invalidates all lines in the selected ways.</li> <li>Writes to this register are enabled when the access bit for the processor is set in the SCU Access Control Register. See <a href="#">SCU Access Control Register on page 9-13</a>.</li> </ul>
<b>Configurations</b>	Available in all configurations.
<b>Attributes</b>	See the register summary in <a href="#">Table 9-2 on page 9-5</a> .

Figure 9-5 shows the SCU Invalidate All Register bit assignments.



**Figure 9-5 SCU Invalidate All Register bit assignments**

Table 9-6 shows the SCU Invalidate All Register bit assignments.

**Table 9-6 SCU Invalidate All Register bit assignments**

Bits	Name	Description
[31:8]	Reserved	SBZ
[7:4]	Processor 1 ways	Specifies the ways that must be invalidated for processor 1. Writing to these bits has no effect if the Cortex-R7 MPCore processor has fewer than two processors.
[3:0]	Processor 0 ways	Specifies the ways that must be invalidated for processor 0.



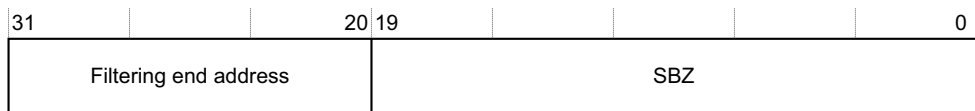


Figure 9-7 Master Filtering End Address Register bit assignments

Table 9-8 shows the Master Filtering End Address Register bit assignments.

Table 9-8 Master Filtering End Address Register bit assignments

Bits	Name	Description
[31:20]	Filtering end address	End address for use with master port 1 in a two-master port configuration, when address filtering is enabled. The default value is the value of <b>MFILTEREND</b> sampled on exit from reset. The value on the input gives the upper address bits with 1MB granularity.
[19:0]	Reserved	SBZ

See [Configuration signals on page A-7](#). See also [AXI master port 1 on page 2-19](#)

### 9.3.7 Peripherals Filtering Start Address Register

The Peripherals Filtering Start Address Register characteristics are:

<b>Purpose</b>	Provides the filtering start address for the peripheral port.
<b>Usage constraints</b>	This register is read-only. For the peripheral port region to operate, the filtering start address must be lower than the filtering end address.
<b>Configurations</b>	Available in all configurations.
<b>Attributes</b>	See the register summary in <a href="#">Table 9-2 on page 9-5</a> .

Figure 9-8 shows the Peripherals Filtering Start Address Register bit assignments.

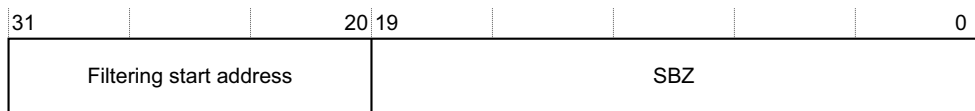


Figure 9-8 Peripherals Filtering Start Address Register bit assignments

Table 9-9 shows the Peripherals Filtering Start Address Register bit assignments.

Table 9-9 Peripherals Filtering Start Address Register bit assignments

Bits	Name	Description
[31:20]	Filtering start address	Filtering start address for the peripheral port. The default value is the value of <b>PFILTERSTART</b> sampled on exit from reset. The value on the input gives the upper address bits with 1MB granularity.
[19:0]	Reserved	SBZ

See [AXI peripheral port on page 2-20](#).

### 9.3.8 Peripherals Filtering End Address Register

The Peripherals Filtering End Address Register characteristics are:

<b>Purpose</b>	Provides the filtering end address for the peripheral port.
<b>Usage constraints</b>	This register is read-only. It has an inclusive address as its end address. This means that the topmost megabyte of address space of memory can be included in the filtering address range. For the peripheral port region to operate, the filtering start address must be lower than the filtering end address.
<b>Configurations</b>	Available in all configurations.
<b>Attributes</b>	See the register summary in <a href="#">Table 9-2 on page 9-5</a> .

[Figure 9-9](#) shows the Peripherals Filtering End Address Register bit assignments.



**Figure 9-9 Peripherals Filtering End Address Register bit assignments**

[Table 9-10](#) shows the Peripherals Filtering End Address Register bit assignments.

**Table 9-10 Peripherals Filtering End Address Register bit assignments**

Bits	Name	Description
[31:20]	Filtering end address	Filtering end address for the peripheral port. The default value is the value of <b>PFILTEREND</b> sampled on exit from reset. The value on the input gives the upper address bits with 1MB granularity.
[19:0]	Reserved	SBZ

See [Configuration signals on page A-7](#). See also [AXI peripheral port on page 2-20](#).

### 9.3.9 SCU Access Control Register

The SCU Access Control Register characteristics are:

<b>Purpose</b>	Controls access to the following registers on a per processor basis: <ul style="list-style-type: none"> <li>• <a href="#">SCU Control Register on page 9-6</a>.</li> <li>• <a href="#">SCU CPU Power Status Register on page 9-9</a>.</li> <li>• <a href="#">SCU Invalidate All Register on page 9-10</a>.</li> <li>• <a href="#">SCU Error Bank First Entry Register on page 9-14</a>.</li> <li>• <a href="#">SCU Error Bank Second Entry Register on page 9-15</a>.</li> </ul>
<b>Usage constraints</b>	There are no usage constraints.
<b>Configurations</b>	Available in all configurations.
<b>Attributes</b>	See the register summary in <a href="#">Table 9-2 on page 9-5</a> .

[Figure 9-10 on page 9-14](#) shows the SCU Access Control Register bit assignments.

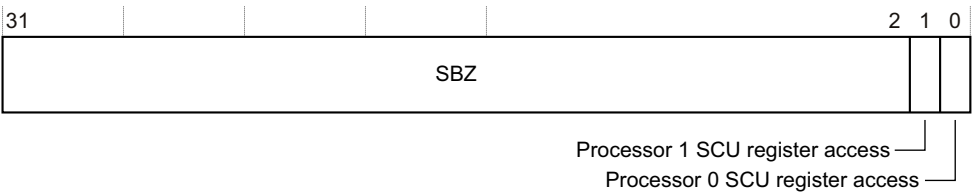


Figure 9-10 SCU Access Control Register bit assignments

Table 9-11 shows the SCU Access Control Register bit assignments.

Table 9-11 SCU Access Control Register bit assignments

Bits	Name	Description
[31:2]	Reserved	SBZ
[1]	Processor 1 SCU register access	1 Processor 1 can access the SCU registers. This is the default. 0 Processor 1 cannot access the SCU registers.
[0]	Processor 0 SCU register access	1 Processor 0 can access the SCU registers. This is the default. 0 Processor 0 cannot access the SCU registers.

9.3.10 SCU Error Bank First Entry Register

The SCU Error Bank First Entry Register characteristics are:

- Purpose** Shows the first SCU error bank entry.
- Usage constraints** There are no usage constraints.
- Configurations** Available only in configurations where ECC is implemented.
- Attributes** See the register summary in Table 9-2 on page 9-5.

Figure 9-11 shows the SCU Error Bank First Entry Register bit assignments.

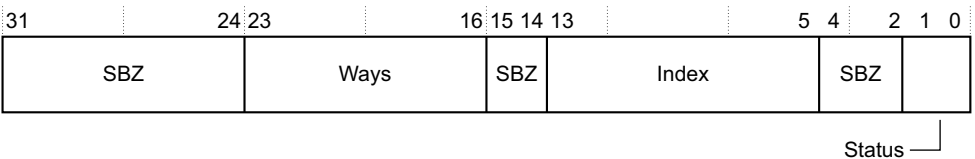


Figure 9-11 SCU Error Bank First Entry Register bit assignments

Table 9-12 shows the SCU Error Bank First Entry Register bit assignments.

Table 9-12 SCU Error Bank First Entry Register bit assignments

Bits	Name	Function
[31:24]	Reserved	SBZ
[23:16]	Ways	Ways in the SCU tag ram, four bits per processor.
[15:14]	Reserved	SBZ

**Table 9-12 SCU Error Bank First Entry Register bit assignments (continued)**

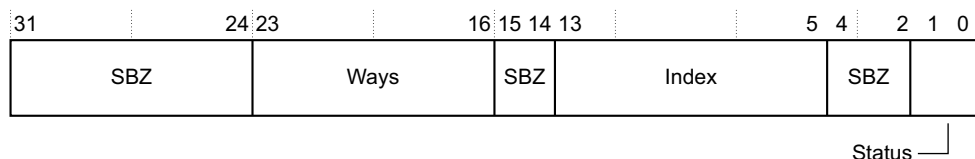
Bits	Name	Function
[13:5]	Index	Index in the SCU tag RAM.
[4:2]	Reserved	SBZ
[1:0]	Status	Error status. The values are: 0b00          No error. 0b01          Error seen by SCU tag RAM, but not handled by processor. 0b10          Error seen by both SCU and processor. 0b11          Error is confirmed by software.

### 9.3.11 SCU Error Bank Second Entry Register

The SCU Error Bank Second Entry Register characteristics are:

- Purpose** Shows the second SCU error bank entry.
- Usage constraints** There are no usage constraints.
- Configurations** Available only in configurations where ECC is implemented.
- Attributes** See the register summary in [Table 9-2 on page 9-5](#).

[Figure 9-12](#) shows the SCU Error Bank Second Entry Register bit assignments.

**Figure 9-12 SCU Error Bank Second Entry Register bit assignments**

[Table 9-13](#) shows the SCU Error Bank Second Entry Register bit assignments.

**Table 9-13 SCU Error Bank Second Entry Register bit assignments**

Bits	Name	Function
[31:24]	Reserved	SBZ
[23:16]	Ways	Ways in the SCU tag ram, four bits per processor.
[15:14]	Reserved	SBZ
[13:5]	Index	Index in the SCU tag RAM.
[4:2]	Reserved	SBZ
[1:0]	Status	Error status. The values are: 0b00          No error. 0b01          Error seen by SCU tag RAM, but not handled by processor. 0b10          Error seen by both SCU and processor. 0b11          Error is confirmed by software.

### 9.3.12 SCU Debug tag RAM access

You can access a specific faulty location in a Tag RAM or inject fake errors to check the ECC mechanism in the SCU. Three SCU registers are provided:

- [SCU Debug Tag RAM Operation Register](#) to select the type of operation and the selected Tag RAM.
- [SCU Debug Tag RAM Data Value Register on page 9-17](#) to select a specific Tag value.
- [SCU Debug Tag RAM ECC Chunk Register on page 9-18](#) to select the ECC chunk associated with the Tag value.

#### Accessing an SCU tag RAM location

To read a given SCU tag RAM location:

1. Write the [SCU Debug Tag RAM Operation Register](#).
2. Read the [SCU Debug Tag RAM Data Value Register on page 9-17](#) or the [SCU Debug Tag RAM ECC Chunk Register on page 9-18](#).

To write a given SCU tag RAM location:

1. Write the [SCU Debug Tag RAM Data Value Register on page 9-17](#) or the [SCU Debug Tag RAM ECC Chunk Register on page 9-18](#).
2. Write the [SCU Debug Tag RAM Operation Register](#).

#### SCU Debug Tag RAM Operation Register

The SCU Debug Tag RAM Operation Register characteristics are:

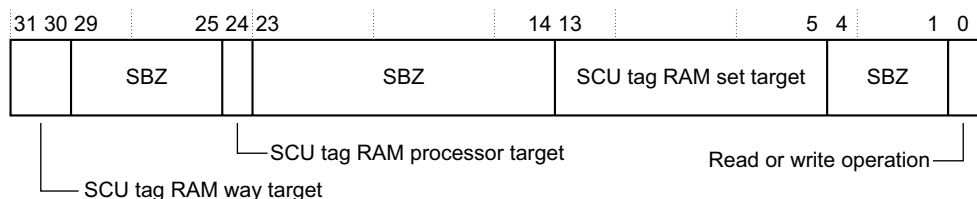
**Purpose** Gives the address and action for SCU tag RAM direct access.

**Usage constraints** There are no usage constraints.

**Configurations** Available in all configurations.

**Attributes** See the register summary in [Table 9-2 on page 9-5](#).

[Figure 9-13](#) shows the SCU Debug Tag RAM Operation Register bit assignments.



**Figure 9-13 SCU Debug Tag RAM Operation Register bit assignments**

Table 9-14 shows the SCU Debug Tag RAM Operation Register bit assignments.

**Table 9-14 SCU Debug Tag RAM Operation Register bit assignments**

Bits	Name	Function
[31:30]	SCU tag RAM way target	Indicates the number of the RAM way.
[29:25]	Reserved	SBZ
[24]	SCU tag RAM processor target	Indicates the processor target: 0 Processor 0. 1 Processor 1.
[23:14]	Reserved	SBZ
[13:5]	SCU tag RAM set target	Index to read or write the SCU tag RAM.
[4:1]	Reserved	SBZ
[0]	Read or write operation	Specifies whether it is a read or write operation: 0 Read. 1 Write.

### SCU Debug Tag RAM Data Value Register

The SCU Debug Tag RAM Data Value Register characteristics are:

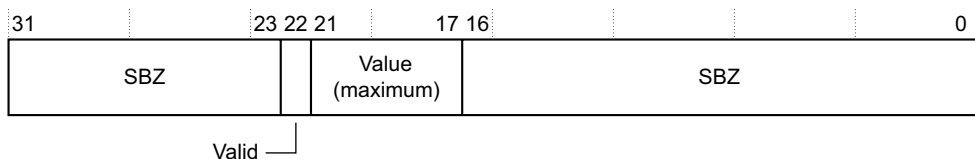
**Purpose** Gives the data value for SCU tag RAM direct access.

**Usage constraints** There are no usage constraints.

**Configurations** Available in all configurations.

**Attributes** See the register summary in [Table 9-2 on page 9-5](#).

Figure 9-14 shows the SCU Debug Tag RAM Data Value Register bit assignments.



**Figure 9-14 SCU Debug Tag RAM Data Value Register bit assignments**



Table 9-15 shows the SCU Debug Tag RAM Data Value Register bit assignments.

**Table 9-15 SCU Debug Tag RAM Data Value Register bit assignments**

Bits	Name	Function
[31:23]	Reserved	SBZ
[22]	Valid	Valid bit.
[21:17]	Value	Tag value: <b>[21]</b> 4K. <b>[21:20]</b> 8K. <b>[21:19]</b> 16K. <b>[21:18]</b> 32K. <b>[21:17]</b> 64K. Unused bits are Reserved.
[16:0]	Reserved	SBZ

### SCU Debug Tag RAM ECC Chunk Register

The SCU Debug Tag RAM ECC Chunk Register characteristics are:

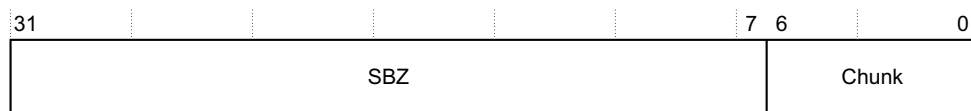
<b>Purpose</b>	Shows the ECC chunk value.
----------------	----------------------------

**Usage constraints** There are no usage constraints.

<b>Configurations</b>	Available only in configurations where ECC is implemented.
-----------------------	--

**Attributes** See the register summary in [Table 9-2 on page 9-5](#).

Figure 9-15 shows the SCU Debug Tag RAM ECC Chunk Register bit assignments.



**Figure 9-15 SCU Debug Tag RAM ECC Chunk Register bit assignments**

Table 9-16 shows the SCU Debug Tag RAM ECC Chunk Register bit assignments.

### Table 9-16 SCU Debug Tag RAM ECC Chunk Register bit assignments

Bits	Name	Function
[31:7]	-	SBZ
[6:0]	Chunk	ECC chunk value

## 9.4 Interrupt controller

The interrupt controller is compliant with the ARM *Generic Interrupt Controller (GIC) v1.0* architecture. This section describes the implementation-defined features of the interrupt controller, and does not reproduce information already in the *ARM® Generic Interrupt Controller Architecture Specification*. This section describes the following:

- [About the interrupt controller.](#)
- [Distributor register descriptions on page 9-20.](#)
- [Interrupt interface register descriptions on page 9-27.](#)

### 9.4.1 About the interrupt controller

The interrupt controller is a single functional unit that is located in a Cortex-R7 MPCore processor design. There is one interrupt interface per processor in the design. This implementation of the interrupt controller does not support the Security Extensions.

The interrupt controller is memory-mapped. The Cortex-R7 processors access it by using a private interface through the SCU. See [Private memory region on page 2-23](#).

#### Interrupt controller clock frequency

The interrupt controller runs on **PERIPHCLK**. The clock period is configured, during integration, as an integer division of the Cortex-R7 MPCore processor clock (**CLK**) period. This division, *N*, must be greater than or equal to two. As a consequence, the minimum pulse width of signals driving external interrupt lines is *N CLK* cycles. See [Clocking, resets, and initialization on page 2-5](#) for a description of **PERIPHCLK** and **PERIPHCLKEN**.

The timers and watchdogs use the same clock as the interrupt controller.

#### Interrupt distributor interrupt sources

The interrupt distributor centralizes all interrupt sources before dispatching the highest priority ones to each Cortex-R7 processor.

The Cortex-R7 MPCore processor supports only the 1-*N* service model.

All interrupt sources are identified by a unique ID. All interrupt sources have their own configurable priority and list of targeted Cortex-R7 processors. This is a list of processors that the interrupt is sent to when triggered by the interrupt distributor.

Interrupt sources are of the following types:

#### Software Generated Interrupts (SGI)

Each Cortex-R7 processor has private interrupts, ID0-ID15, that can only be triggered by software. These interrupts are aliased so that there is no requirement for a requesting processor to determine its own processor ID when it deals with SGIs. The priority of an SGI depends on the value set by the receiving processor in the banked SGI priority registers, not the priority set by the sending processor.

#### Global timer, PPI[0]

The global timer uses ID27. See [Global timer on page 9-35](#).

#### A legacy nFIQ input, PPI[1]

In the Cortex-R7 processor, the **nFIQ** input is connected both to the corresponding processor and to the GIC. If the interrupt controller is enabled, **nFIQ** is mapped to PPI[1], and can still cause an IRQ exception.

When a Cortex-R7 processor uses the interrupt controller, rather than the legacy input in the legacy mode, by enabling its own processor interface, the legacy **nFIQ** input is also treated like other interrupt lines and uses ID28. In this case, you can mask the legacy FIQ interrupt by setting bit[6] of the CPSR, the F bit, so that only the interrupt line of the GIC is used.

#### Private timer, PPI[2]

Each Cortex-R7 processor has its own private timers that can generate interrupts, using ID29. See [Private timer and watchdog on page 9-29](#).

#### Watchdog timers, PPI[3]

Each Cortex-R7 processor has its own watchdog timers that can generate interrupts, using ID30. See [Private timer and watchdog on page 9-29](#).

#### A legacy nIRQ input, PPI[4]

In legacy IRQ mode the legacy **nIRQ** input, on a per processor basis, bypasses the interrupt distributor logic and directly drives interrupt requests into the Cortex-R7 processor.

When a Cortex-R7 processor uses the interrupt controller, rather than the legacy input in the legacy mode, by enabling its own processor interface, the legacy **nIRQ** input is treated like other interrupt lines and uses ID31.

#### Shared Peripheral Interrupts (SPI)

SPIs are triggered by events generated on associated interrupt input lines. The interrupt controller can support up to 480 interrupt input lines. The interrupt input lines can be configured to be edge sensitive (posedge) or level sensitive (high level). SPIs start at ID32. The **IRQS** bus generates SPIs.

#### ———— Note ————

The Cortex-R7 MPCore processor does not provide internal synchronization for the interrupt signals, **nIRQ**, **nFIQ**, and **IRQS**.

#### Priority formats

The Cortex-R7 MPCore processor implements a four-bit version of the priority format in the *ARM® Generic Interrupt Controller Architecture Specification*.

### 9.4.2 Distributor register descriptions

This section describes the registers that the distributor provides. [Table 9-17 on page 9-21](#) shows the distributor registers.

Registers not described in [Table 9-17 on page 9-21](#) are RAZ/WI. This section does not reproduce information about registers already described in the *ARM® Generic Interrupt Controller Architecture Specification 1.0*.

The ICDIPR and ICDIPTR registers are byte accessible and word accessible. All other registers in [Table 9-17 on page 9-21](#) are word accessible. Any other access is UNPREDICTABLE.

See [Private memory region on page 2-23](#) for the offset of this page from PERIPBASE[31:13].

Table 9-17 Distributor register summary

Base	Name	Type	Reset	Width	Description
0x000	ICDDCR	RW	0x00000000	32	<a href="#">Distributor Control Register on page 9-22</a>
0x004	ICDICTR	RO	Configuration dependent	32	<a href="#">Interrupt Controller Type Register on page 9-22</a>
0x008	ICDIIDR	RO	0x0300043B	32	<a href="#">Distributor Implementer Identification Register on page 9-24</a>
0x00C - 0x09C	-	-	-	-	Reserved
0x100 - 0x13C	ICDISERn	RW	0x00000000 <sup>a</sup>	32	Interrupt Set-Enable Registers
0x180 - 0x1BC	ICDICERn	RW	0x00000000 <sup>b</sup>	32	Interrupt Clear-Enable Registers
0x200 - 0x23C	ICDISPRn	RW	0x00000000	32	Interrupt Set-Pending Registers
0x280 - 0x2BC	ICDICPRn	RW	0x00000000	32	Interrupt Clear-Pending Registers
0x300 - 0x33C	ICDABRn	RO	0x00000000	32	Active Bit registers
0x380 - 0x3FC	-	-	-	-	Reserved
0x400 - 0x4FC	ICDIPRn	RW	0x00000000	32	Interrupt Priority Registers <sup>c</sup>
0x7FC	-	-	-	-	Reserved
0x800 - 0x9FC	ICDIPTRn	RW <sup>b</sup>	0x00000000	32	Interrupt Processor Targets Registers
0xBFC	-	-	-	-	Reserved
0xC00 - 0xC7C	ICDICFRn	RW	Configuration dependent	32	Interrupt Configuration Registers
0xD00	PPI Status	-	0x00000000	32	<a href="#">PPI Status Register on page 9-24</a>
0xD04 - 0xD3C	SPI Status	RO	0x00000000	32	<a href="#">SPI Status Registers on page 9-25</a>
0xD80 - 0xEFC	-	-	-	-	Reserved
0xF00	ICDSGIR	WO	-	32	Software Generated Interrupt Register
0xF04 - 0xFCC	-	-	-	-	Reserved
0xFD0 - 0xFEC	Peripheral Identification [4:0]	RO	Configuration dependent	8	<a href="#">CoreSight Identification Registers on page 10-5</a>
0xFF0 - 0xFFC	Component Identification [3:0]	RO	-	8	

- a. The reset value for the registers that contain the SGI and PPI interrupts is implementation-dependent.
- b. Not configurable. Reset to 1.
- c. Only the top four bits of each 8-bit field of the register are in use.



Table 9-19 shows the ICDICTR bit assignments.

**Table 9-19 ICDICTR bit assignments**

Bits	Name	Function
[31:11]	Reserved	SBZ
[10]	Reserved	RAZ/WI
[9:8]	Reserved	SBZ
[7:5]	Processor number	The encoding is: 0b000      The configuration contains one Cortex-R7 processor. 0b001      The configuration contains two Cortex-R7 processors. All other values are unused.
[4:0]	IT lines number	The encoding is: 0b00000      The distributor provides 32 interrupts <sup>a</sup> , no external interrupt lines. 0b00001      The distributor provides 64 interrupts, 32 external interrupt lines. 0b00010      The distributor provides 96 interrupts, 64 external interrupt lines. 0b00011      The distributor provides 128 interrupts, 96 external interrupt lines. 0b00100      The distributor provides 160 interrupts, 128 external interrupt lines. 0b00101      The distributor provides 192 interrupts, 160 external interrupt lines. 0b00110      The distributor provides 224 interrupts, 192 external interrupt lines. 0b00111      The distributor provides 256 interrupts, 224 external interrupt lines. 0b01000      The distributor provides 288 interrupts, 256 external interrupt lines. 0b01001      The distributor provides 320 interrupts, 288 external interrupt lines. 0b01010      The distributor provides 352 interrupts, 320 external interrupt lines. 0b01011      The distributor provides 384 interrupts, 352 external interrupt lines. 0b01100      The distributor provides 416 interrupts, 384 external interrupt lines. 0b01101      The distributor provides 448 interrupts, 416 external interrupt lines. 0b01110      The distributor provides 480 interrupts, 448 external interrupt lines. 0b01111      The distributor provides 512 interrupts, 480 external interrupt lines. All other values are unused.

a. The distributor always uses interrupts of IDs 0 to 31 to control any SGIs and PPIs that the interrupt controller might contain.

## Interrupt Processor Targets Registers

This section describes the implementation defined features of the ICDIPTRN. For systems that support only one Cortex-R7 processor, all these registers read as zero, and writes are ignored.

### ———— Note ————

If the Processor Target field is set to 0 for a specific SPI, this interrupt cannot be set pending through the hardware pins or a write to the Set-Pending Register.

## Interrupt Configuration Registers

This section describes the implementation defined features of the ICDICFR. Each bit-pair describes the interrupt configuration for an interrupt. The options for each pair depend on the interrupt type as follows:

**SGI**      The bits are read-only and a bit-pair always reads as 0b10.

**PPI** The bits are read-only:

**PPI[1] and [4]:0b01**

Interrupt is active-LOW level sensitive.

**PPI[0]:0b01**

Interrupt is active-HIGH level sensitive.

**PPI[2] and [3]:0b11**

Interrupt is rising-edge sensitive.

**SPI** The LSB of a bit-pair is read-only and is always 0b1. You can program the MSB of the bit-pair to alter the triggering sensitivity as follows:

0b01 Interrupt is active-HIGH level sensitive.

0b11 Interrupt is rising-edge sensitive.

There are 31 LSPIs, interrupts 32-62.

### Distributor Implementer Identification Register

The ICDIHDR characteristics are:

**Purpose** Provides information about the implementer and the revision of the controller.

**Usage constraints** There are no usage constraints.

**Configurations** Available in all configurations.

**Attributes** See the register summary in [Table 9-17 on page 9-21](#).

[Figure 9-18](#) shows the ICDIHDR bit assignments.

31	24	23	12	11	0
Implementation version			Revision number		Implementer

**Figure 9-18 ICDIHDR bit assignments**

[Table 9-20](#) shows the ICDIHDR bit assignments.

**Table 9-20 ICDIHDR bit assignments**

Bits	Values	Name	Description
[31:24]	0x03	Implementation version	Gives implementation version number.
[23:12]	0x00	Revision number	Returns the revision number of the controller.
[11:0]	0x43B	Implementer	Implementer number.

### PPI Status Register

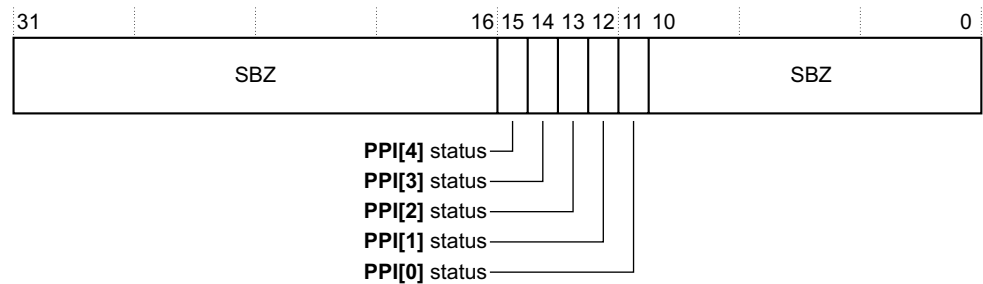
The PPI Status Register characteristics are:

**Purpose** Enables a Cortex-R7 processor to access the status of the inputs on the distributor.

**Usage constraints** A Cortex-R7 processor can only read the status of its own **PPI** and therefore cannot read the status of **PPI** for other Cortex-R7 processors.

- Configurations**
- Available in all configurations.
- Attributes**
- See the register summary in [Table 9-17 on page 9-21](#).

[Figure 9-19](#) shows the PPI Status Register bit assignments.



**Figure 9-19 PPI Status Register bit assignments**

[Table 9-21](#) shows the PPI Status Register bit assignments.

**Table 9-21 PPI Status Register bit assignments**

Bits	Name	Function
[31:16]	Reserved	SBZ
[15:11]	ppi_status	Returns the status of the <b>PPI[4:0]</b> inputs on the distributor: <b>PPI[4]</b> nIRQ. <b>PPI[3]</b> Private watchdog. <b>PPI[2]</b> Private timer. <b>PPI[1]</b> nFIQ. <b>PPI[0]</b> Global timer. PPI[1] and PPI[4] are active LOW PPI[0], PPI[2], and PPI[3] are active HIGH.  <b>Note</b>  These bits return the actual status of the <b>PPI[4:0]</b> signals. The ICDISPRn and ICDICPRn registers can also provide the <b>PPI[4:0]</b> status but because you can write to these registers then they might not contain the actual status of the <b>PPI[4:0]</b> signals.
[10:0]	Reserved	SBZ

**SPI Status Registers**

The SPI Status Register characteristics are:

- Purpose**
- Enables a Cortex-R7 processor to access the status of **IRQS[N:0]** inputs on the distributor.
- Usage constraints**
- There are no usage constraints.
- Configurations**
- Available in all configurations.
- Attributes**
- See the register summary in [Table 9-17 on page 9-21](#).

[Figure 9-20 on page 9-26](#) shows the SPI Status Register bit assignments.



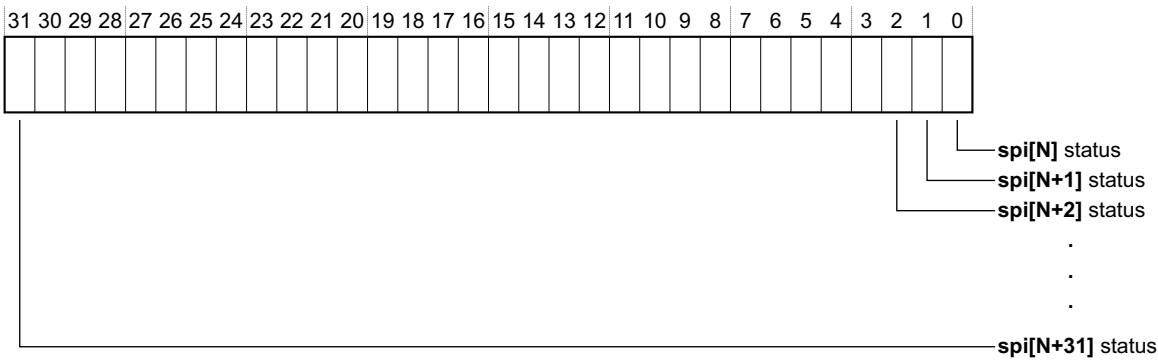


Figure 9-20 SPI Status Register bit assignments

Table 9-22 shows the SPI Status Register bit assignments.

Table 9-22 SPI Status Register bit assignments

Bits	Name	Description
[31:0]	spi_status	Returns the status of the <b>IRQS[N:0]</b> inputs on the distributor: <b>Bit[X] = 0</b> <b>IRQS[X]</b> is LOW. <b>Bit[X] = 1</b> <b>IRQS[X]</b> is HIGH.  ———— <b>Note</b> ————  The <b>IRQS</b> that X refers to depends on its bit position and the base address offset of the SPI Status Register as Figure 9-21 shows.  These bits return the actual status of the <b>IRQS</b> signals. The ICDISPRn and ICDICPRn Registers can also provide the <b>IRQS</b> status but because you can write to these registers then they might not contain the actual status of the <b>IRQS</b> signals.

Figure 9-21 shows the address map that the distributor provides for the SPIs.

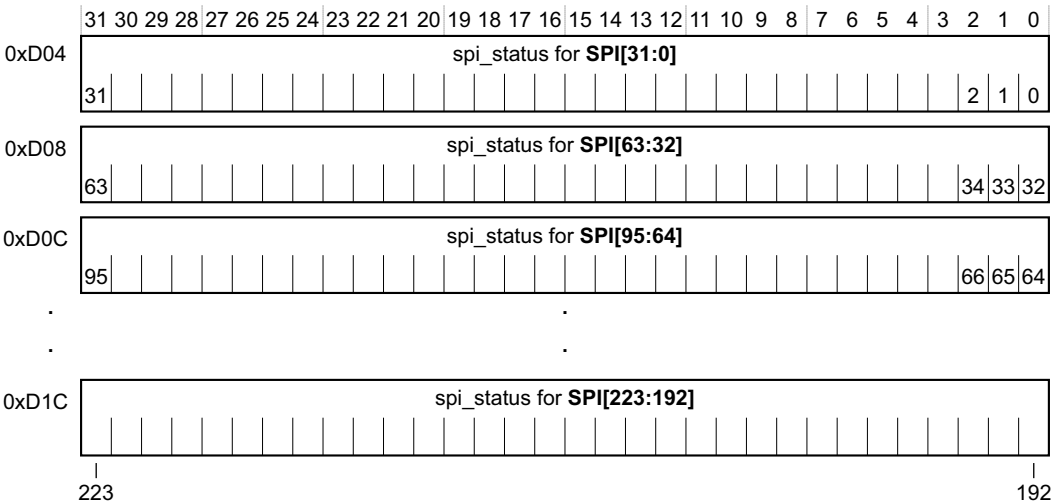


Figure 9-21 SPI Status Register address map

In [Figure 9-21 on page 9-26](#) the values for the SPIs are read-only. This register contains the values for the SPIs for the corresponding Cortex-R7 processor interface. The distributor provides up to seven registers. If you configure the interrupt controller to use fewer than 480 SPIs then it reduces the number of registers accordingly. For locations where interrupts are not implemented, the distributor:

- Ignores writes to the corresponding bits.
- Returns 0 when it reads from these bits.

### 9.4.3 Interrupt interface register descriptions

This section shows the registers that each Cortex-R7 processor interface provides. [Table 9-23](#) shows the Cortex-R7 MPCore processor interface registers. This section does not reproduce information about registers already described in the *ARM® Generic Interrupt Controller Architecture Specification*. These registers are word accessible. Any other access is UNPREDICTABLE.

**Table 9-23 Cortex-R7 processor interface register summary**

Base	Name	Type	Reset	Width	Description
0x000	ICCICR	RW	0x00000000	32	CPU Interface Control Register
0x004	ICCPMR	RW	0x00000000	32	Interrupt Priority Mask Register <sup>a</sup>
0x008	ICCBPR	RW	0x3	32	Binary Point Register
0x00C	ICCIAR	RO	0x000003FF	32	Interrupt Acknowledge Register
0x010	ICCEOIR	WO	-	32	End Of Interrupt Register
0x014	ICCRPR	RO	0x000000FF	32	Running Priority Register
0x018	ICCHPIR	RO	0x000003FF	32	Highest Pending Interrupt Register
0x0FC	ICCIIDR	RO	0x3901243B	32	<i>CPU Interface Implementer Identification Register</i>

a. Only the top four bits of each 8-bit field of the register are in use.

#### CPU Interface Implementer Identification Register

The ICCIIDR Register characteristics are:

<b>Purpose</b>	Provides information about the implementer and the revision of the controller.
<b>Usage constraints</b>	There are no usage constraints.
<b>Configurations</b>	Available in all configurations.
<b>Attributes</b>	See the register summary in <a href="#">Table 9-23</a> .

[Figure 9-22](#) shows the ICCIIDR bit assignments.

31		20	19	16	15	12	11		0
Part number				Architecture number	Revision number	Implementer			

**Figure 9-22 ICCIIDR bit assignments**

[Table 9-24 on page 9-28](#) shows the ICCIIDR bit assignments

**Table 9-24 ICCIHDR bit assignments**

<b>Bits</b>	<b>Values</b>	<b>Name</b>	<b>Description</b>
[31:20]	0x390	Part number	Identifies the peripheral.
[19:16]	0x1	Architecture version	Identifies the architecture version.
[15:12]	0x0	Revision number	Returns the revision number of the interrupt controller. The implementer defines the format of this field.
[11:0]	0x43B	Implementer	<p>Returns the JEP106 code of the company that implemented the Cortex-R7 MPCore processor interface RTL. It uses the following construct:</p> <p>[11:8]           JEP106 continuation code of the implementer.</p> <p>[7]               0.</p> <p>[6:0]            JEP106 code [6:0] of the implementer.</p>

## 9.5 Private timer and watchdog

The private timer and watchdog blocks have the following features:

- A 32-bit counter that generates an interrupt when it reaches zero.
- An eight-bit prescaler value to qualify the clock period.
- Configurable single-shot or auto-reload modes.
- Configurable starting values for the counter.
- The clock for these blocks is **PERIPHCLK**.

The watchdog can be configured as a timer. See [Clocking, resets, and initialization](#) on page 2-5 for a description of **CLK**, **PERIPHCLK**, and **PERIPHCLKEN**.

### 9.5.1 Calculating timer intervals

The timer interval is calculated using the following equation:

$$\left( \frac{(\text{PRESCALER\_value}+1) \times (\text{Load\_value}+1)}{\text{PERIPHCLK}} \right)$$

This equation can be used to calculate the period between two events generated by a timer or watchdog.

### 9.5.2 Private timer and watchdog registers

Addresses are relative to the base address of the timer and watchdog region defined by the private memory map. See [Private memory region](#) on page 2-23. All timer and watchdog registers are word-accessible only. Any other access is UNPREDICTABLE

Use **nPERIPHRESET** to reset these registers, except for the Watchdog Reset Status Register.

**nWDRESET** resets the Watchdog Reset Status Register. See [Private memory region](#) on page 2-23 and [Reset signals](#) on page A-5.

Table 9-25 shows the timer and watchdog registers. All registers not described in Table 9-25 are Reserved.

**Table 9-25 Timer and watchdog registers**

Offset	Type	Reset Value	Description
0x00	RW	0x00000000	<a href="#">Private Timer Load Register</a> on page 9-30
0x04	RW	0x00000000	<a href="#">Private Timer Counter Register</a> on page 9-30
0x08	RW	0x00000000	<a href="#">Private Timer Control Register</a> on page 9-30
0x0C	RW	0x00000000	<a href="#">Private Timer Interrupt Status Register</a> on page 9-31
0x20	RW	0x00000000	<a href="#">Watchdog Load Register</a> on page 9-31
0x24	RW	0x00000000	<a href="#">Watchdog Counter Register</a> on page 9-31
0x28	RW	0x00000000	<a href="#">Watchdog Control Register</a> on page 9-32
0x2C	RW	0x00000000	<a href="#">Watchdog Interrupt Status Register</a> on page 9-33
0x30	RW	0x00000000	<a href="#">Watchdog Reset Status Register</a> on page 9-34
0x34	WO	-	<a href="#">Watchdog Disable Register</a> on page 9-34

**Note**

The private timers stop counting when the associated processor is in debug state.

**Private Timer Load Register**

The Timer Load Register contains the value copied to the Timer Counter Register when it decrements down to zero with auto-reload mode enabled. Writing to the Timer Load Register means that you also write to the Timer Counter Register.

**Private Timer Counter Register**

The Timer Counter Register is a decrementing counter.

The Timer Counter Register decrements if the timer is enabled using the timer enable bit in the Timer Control Register. If a Cortex-R7 processor timer is in debug state, the counter only decrements when the processor returns to non-debug state.

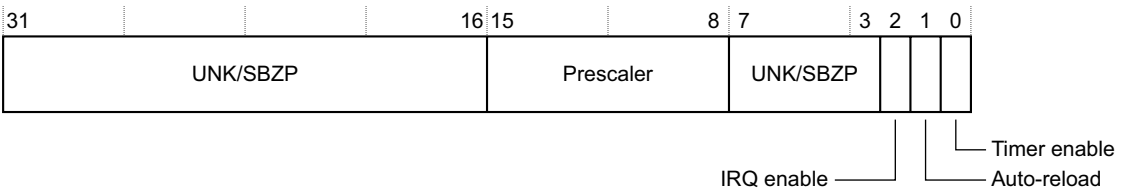
When the Timer Counter Register reaches zero and auto-reload mode is enabled, it reloads the value in the Timer Load Register and then decrements from that value. If auto-reload mode is not enabled, the Timer Counter Register decrements down to zero and stops.

When the Timer Counter Register reaches zero, the timer interrupt status event flag is set and the interrupt ID 29 is set as pending in the Interrupt Distributor, if interrupt generation is enabled in the Timer Control Register.

Writing to the Timer Counter Register or Timer Load Register forces the Timer Counter Register to decrement from the newly written value.

**Private Timer Control Register**

Figure 9-23 shows the Private Timer Control Register bit assignments.



**Figure 9-23 Private Timer Control Register bit assignments**

Table 9-26 shows the Private Timer Control Register bit assignments.

**Table 9-26 Private Timer Control Register bit assignments**

Bits	Name	Function
[31:16]	-	UNK/SBZP.
[15:8]	Prescaler	The prescaler modifies the clock period for the decrementing event for the Counter Register. See <a href="#">Calculating timer intervals on page 9-29</a> for the equation.
[7:3]	-	UNK/SBZP.

**Table 9-26 Private Timer Control Register bit assignments (continued)**

Bits	Name	Function
[2]	IRQ enable	If set, the interrupt ID 29 is set as pending in the Interrupt Distributor when the event flag is set in the Timer Status Register.
[1]	Auto-reload	Auto-reload enable: 0 Single-shot mode. Counter decrements down to zero, sets the event flag and stops. 1 Auto-reload mode. Each time the Counter Register reaches zero, it is reloaded with the value contained in the Timer Load Register.
[0]	Timer enable	Timer enable: 0 Timer is disabled and the counter does not decrement. All registers can still be read and written. 1 Timer is enabled and the counter decrements normally.

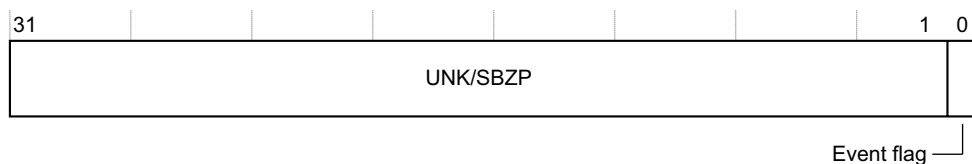
The timer is incremented every prescaler value plus 1. For example, if the prescaler has a value of five, the global timer is incremented every six clock cycles. **PERIPCLK** is the reference clock for this.

### Private Timer Interrupt Status Register

Figure 9-24 shows the Private Timer Interrupt Status Register bit assignments.

This is a banked register for all Cortex-R7 processors present.

The event flag is a sticky bit that is automatically set when the Counter Register reaches zero. If the timer interrupt is enabled, Interrupt ID 29 is set as pending in the Interrupt Distributor after the event flag is set. The event flag is cleared by writing a 1 to bit[0].

**Figure 9-24 Private Timer Interrupt Status Register bit assignments**

### Watchdog Load Register

The Watchdog Load Register contains the value copied to the Watchdog Counter Register when it decrements down to zero with auto-reload mode enabled, in Timer mode. Writing to the Watchdog Load Register means that you also write to the Watchdog Counter Register.

### Watchdog Counter Register

The Watchdog Counter Register is a decrementing counter.

It decrements if the Watchdog is enabled using the Watchdog enable bit in the Watchdog Control Register. If the Cortex-R7 processor associated with the Watchdog is in debug state, the counter does not decrement until the processor returns to non-debug state.

When the Watchdog Counter Register reaches zero and auto-reload mode is enabled, and in timer mode, it reloads the value in the Watchdog Load Register and then decrements from that value. If auto-reload mode is not enabled or the watchdog is not in timer mode, the Watchdog Counter Register decrements down to zero and stops.

When in watchdog mode the only way to update the Watchdog Counter Register is to write to the Watchdog Load Register. When in timer mode the Watchdog Counter Register is write accessible.

The behavior of the watchdog when the Watchdog Counter Register reaches zero depends on its current mode:

**Timer mode** When the Watchdog Counter Register reaches zero, the watchdog interrupt status event flag is set. If interrupt generation is enabled in the Watchdog Control Register, interrupt ID 30 is set as pending in the Interrupt Distributor.

**Watchdog mode**

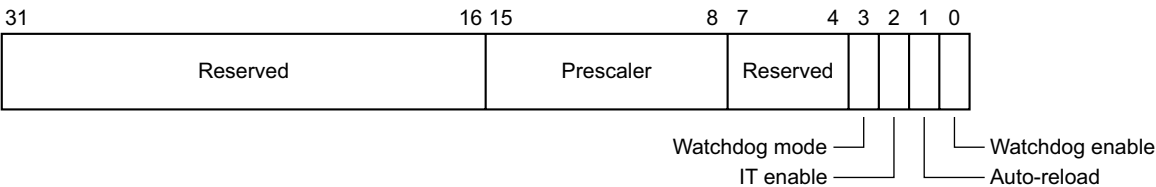
If a software failure prevents the Watchdog Counter Register from being refreshed:

- The Watchdog Counter Register reaches zero.
- The Watchdog reset status flag is set.
- The associated **WDRESETREQ** reset request output is asserted.  
**WDRESETREQ** is valid for one **PERIPHCLK** cycle.

The external reset source is then responsible for resetting all or part of the Cortex-R7 MPCore processor design.

**Watchdog Control Register**

Figure 9-25 shows the Watchdog Control Register bit assignments.



**Figure 9-25 Watchdog Control Register bit assignments**

Table 9-27 shows the Watchdog Control Register bit assignments.

**Table 9-27 Watchdog Control Register bit assignments**

Bits	Name	Function
[31:16]	-	Reserved.
[15:8]	Prescaler	The prescaler modifies the clock period for the decrementing event for the Counter Register. See <a href="#">Calculating timer intervals on page 9-29</a> .
[7:4]	-	Reserved.
[3]	Watchdog mode	Watchdog mode enable: 0 Timer mode. This is the default. Writing a zero to this bit has no effect. You must use the Watchdog Disable Register to put the watchdog into timer mode. See <a href="#">Watchdog Disable Register on page 9-34</a> . 1 Watchdog mode.

Table 9-27 Watchdog Control Register bit assignments (continued)

Bits	Name	Function
[2]	IT Enable	If set, the interrupt ID 30 is set as pending in the Interrupt Distributor when the event flag is set in the watchdog Status Register. In watchdog mode this bit is ignored.
[1]	Auto-reload	Auto-reload enable: 0 Single-shot mode. Counter decrements down to zero, sets the event flag and stops. 1 Auto-reload mode. Each time the Counter Register reaches zero, it is reloaded with the value contained in the Load Register and then continues decrementing.
[0]	Watchdog Enable	Global watchdog enable: 0 Watchdog is disabled and the counter does not decrement. All registers can still be read or written. 1 Watchdog is enabled and the counter decrements normally.

Watchdog Interrupt Status Register

Figure 9-26 shows the Watchdog Interrupt Status Register bit assignments.

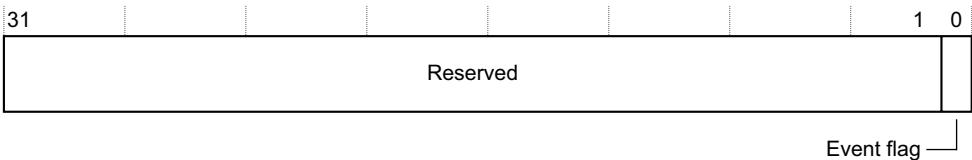


Figure 9-26 Watchdog Interrupt Status Register bit assignments

The event flag is a sticky bit that is automatically set when the Counter Register reaches zero in timer mode. If the watchdog interrupt is enabled, Interrupt ID 30 is set as pending in the Interrupt Distributor after the event flag is set. The event flag is cleared when written with a value of 1. Trying to write a zero to the event flag or a one when it is not set has no effect.





## 9.6 Global timer

The global timer has the following features:

- The global timer is a 64-bit incrementing counter with an auto-incrementing feature. It continues incrementing after sending interrupts.
- The global timer is memory-mapped in the private memory region. See [Private memory region on page 2-23](#).
- The global timer is accessible to all Cortex-R7 processors in the Cortex-R7 MPCore design. Each processor has a private 64-bit comparator that is used to assert a private interrupt when the global timer has reached the comparator value. All the Cortex-R7 processors in a design use the banked ID, ID27, for this interrupt. ID27 is sent to the interrupt controller as a Private Peripheral Interrupt. See [Distributor register descriptions on page 9-20](#).
- Each processor can only access its own comparator registers. It cannot access the comparator of another processor.
- The interrupt from a comparator only goes to the associated processor. A processor cannot see the interrupt from the comparator of another processor.
- The global timer is clocked by **PERIPHCLK**.

---

**Note**

- The comparators for each processor with the global timer fire when the timer value is greater than or equal to the value you have programmed.
  - The global timer does not stop counting when any of the processors are in debug state.
- 

### 9.6.1 Global timer registers

[Table 9-28](#) shows the global timer registers. The offset is relative to PERIPH\_BASE\_ADDR + 0x0200. Use **nPERIPHRESET** to reset these registers.

**Table 9-28 Global timer registers**

Offset	Type	Reset value	Description
0x00	RW	0x00000000	<a href="#">Global Timer Counter Registers</a>
0x04	RW	0x00000000	
0x08	RW	0x00000000	<a href="#">Global Timer Control Register on page 9-36</a>
0x0C	RW	0x00000000	<a href="#">Global Timer Interrupt Status Register on page 9-37</a>
0x10	RW	0x00000000	<a href="#">Comparator Value Registers on page 9-37</a>
0x14	RW	0x00000000	
0x18	RW	0x00000000	<a href="#">Auto-increment Register on page 9-37</a>

#### Global Timer Counter Registers

There are two timer counter registers. They are the lower 32-bit timer counter at offset 0x00 and the upper 32-bit timer counter at offset 0x04.

You must access these registers with 32-bit accesses. You cannot use STRD/LDRD. Any other access is UNPREDICTABLE.

To modify the register proceed as follows:

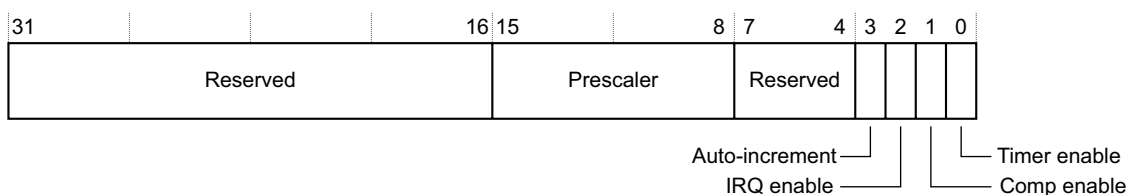
1. Clear the timer enable bit in the Global Timer Control Register.
2. Write the lower 32-bit timer counter register.
3. Write the upper 32-bit timer counter register.
4. Set the timer enable bit.

To get the value from the Global Timer Counter register proceed as follows:

1. Read the upper 32-bit timer counter register.
2. Read the lower 32-bit timer counter register.
3. Read the upper 32-bit timer counter register again. If the value is different to the 32-bit upper value read previously, go back to step 2. Otherwise the 64-bit timer counter value is correct.

### Global Timer Control Register

Figure 9-28 shows the Global Timer Control Register bit assignments.



**Figure 9-28 Global Timer Control Register bit assignments**

Table 9-29 shows the Global Timer Control Register bit assignments.

**Table 9-29 Global Timer Control Register bit assignments**

Bits	Name	Function
[31:16]	-	Reserved
[15:8]	Prescaler	The prescaler modifies the clock period for the decrementing event for the Counter Register. See <a href="#">Calculating timer intervals on page 9-29</a> for the equation.
[7:4]	-	Reserved
[3]	Auto-increment <sup>a</sup>	<p>This bit is banked per Cortex-R7 processor:</p> <p>0 Single shot mode. When the counter reaches the comparator value, sets the event flag. It is the responsibility of software to update the comparator value to get more events.</p> <p>1 Auto-increment mode. Each time the counter reaches the comparator value, the comparator register is incremented with the auto-increment register, so that more events can be set periodically without any software updates.</p>

## Global Timer Interrupt Status Register

a. When the Auto-increment and Comp enable bits are set, an IRQ is generated every auto-increment register value.

## Global Timer Interrupt Status Register

The event flag is a sticky bit that is automatically set when the Counter Register reaches the Comparator Register value. If the timer interrupt is enabled, Interrupt ID 27 is set as pending in the Interrupt Distributor after the event flag is set. The event flag is cleared when written to 1. [Figure 9-29](#) shows the Global Timer Interrupt Status Register bit assignments.



## Comparator Value Registers

You must access these registers with 32-bit accesses. You cannot use STRD/LDRD. There is a Comparator Value Register for each Cortex-R7 processor.

1. Clear the Comp Enable bit in the Timer Control Register.
2. Write the lower 32-bit Comparator Value Register.
3. Write the upper 32-bit Comparator Value Register.
4. Set the Comp Enable bit and, if necessary, the IRQ enable bit.

## Auto-increment Register

This 32-bit register gives the increment value of the Comparator Register when the Auto-increment bit is set in the Timer Control Register. Each Cortex-R7 processor present has its own Auto-increment Register.

If the comp enable and auto-increment bits are set when the global counter reaches the Comparator Register value, the comparator is incremented by the auto-increment value, so that a new event can be set periodically.

The global timer is not affected, and continues incrementing.

## 9.7 Accelerator Coherency Port

The following sections describe the ACP:

- [Coherent and noncoherent mode](#).
- [Read accesses in coherent mode](#).
- [Write accesses in coherent mode](#).
- [AXI protocol configurability, xID and AxUSER](#).
- [AXI protocol restrictions on page 9-40](#).
- [ACP bridge on page 9-40](#).

### 9.7.1 Coherent and noncoherent mode

The coherency between the ACP traffic and the L1 data cache of the processors is triggered when **AxUSER[0]** = 1 and **AxCACHE[1]** = 1:

- **AxUSER[0]** is the shared bit.
- **AxCACHE[1]** = 1 indicates it is a NC, WT, WB, or a WBWA memory region.
- **AxCACHE[1]** = 0 indicates it is a SO, or DV memory region. Using the **AxCACHE[1]** bit prevents SO and DV memory regions from taking part in the coherency mechanism, that is, SCU lookups, on the ACP.

x represents either R for read or W for write.

### 9.7.2 Read accesses in coherent mode

Any read access from the ACP in coherent mode behaves as follows:

- If the data is present in the L1 data cache of either processor, the data is returned from the L1 data cache of the processor that has the data.
- If the data is not present in the L1 data cache of either processor, the data is returned from the L2 memory.

### 9.7.3 Write accesses in coherent mode

Any write access from the ACP in coherent mode behaves as follows:

- If the data is present in the L1 data cache of either processor, and:
  - If the cache line is clean, the line is invalidated, and the write is done on the L2 memory.
  - If the cache line is dirty, the line is clean and invalidated, and the write is done on the L2 memory.
- If the data is not present in the L1 data cache of either processor, the data is written to the L2 memory.

### 9.7.4 AXI protocol configurability, xID and AxUSER

The **xIDSC** bus width is configurable and must be greater than or equal to 4.

Table 9-30 shows the **AWUSERSC[5:0]** encoding for the ACP.

**Table 9-30 AWUSERSC encoding for ACP**

AWUSERSC value	Information propagated on AXI master port 0, AXI master port 1, and peripheral port	Usage
[0]	Yes	See <i>Coherent and noncoherent mode</i> on page 9-39.
[4:1]	Yes	Ignored by the SCU.
[5]	No	SBZ if byte line strobes are sparse. This is the default. SBO if all byte line strobes are set according to the AXI access rules.

Table 9-31 shows the **ARUSERSC[4:0]** encoding for the ACP.

**Table 9-31 ARUSERSC encoding for ACP**

ARUSERSC value	Information propagated on AXI master port 0, AXI master port 1, and peripheral port	Usage
[0]	Yes	See <i>Coherent and noncoherent mode</i> on page 9-39.
[4:1]	Yes	Ignored by the SCU.

### 9.7.5 AXI protocol restrictions

The ACP is AXI compliant, but not does support the full protocol:

- The **APROT[1]** input is ignored and the **APROT[1]** output is always LOW on the AXI master buses and the AXI peripheral bus.
- Only swap-like accesses are accepted, that is, a locked read followed by a non-locked write. **AWLOCKSC[1]** is not forwarded from the ACP to any master ports on **AWLOCKM0[1]** or **AWLOCKM1[1]**. These ports are always LOW.
- The ACP does not support exclusive accesses, and there is no exclusive monitor. If either master attempts an exclusive read, the ACP returns an OKAY response instead of an EXOKAY response. The master can treat this as an error condition, indicating that the exclusive access is not supported. ARM recommends that the master does not perform the write portion of this exclusive operation.

### 9.7.6 ACP bridge

The ACP contains an optional ACP bridge. This bridge has an optional ECC support and has an impact on the performance.

With the ACP bridge:

- The ACP is ECC protected if ECC is present on the bus. See *ECC on external AXI bus* on page 7-7.
- All types of AXI transfer are supported.
- There are no special timing recommendations between the AXI address channel and the AXI data channel.

Without the ACP bridge

- The ACP is not ECC protected.

- Only the following accesses are supported:
  - For 32 bytes, **AxLEN** = 0x3, **AxSIZE** = 0x3, **AxBURST** = 01 (incr), and **AxADDR[4:0]** = 00000.
  - For 32 bytes, **AxLEN** = 0x3, **AxSIZE** = 0x3, **AxBURST** = 10 (wrap), and **AxADDR[2:0]** = 000).
  - For 16 bytes, **AxLEN** = 0x1, **AxSIZE** = 0x3, **AxBURST** = 01 (incr), and **AxADDR[3:0]** = 0000).
  - For 8 bytes: **AxLEN** = 0x0, **AxSIZE** = 0x3, **AxBURST** = any, and **AxADDR** = any.

**x** represents either R for read or W for write.  
All other access types generate a slave error.
- A write access can be accepted every four cycles.
- The addresses must be sent at least six cycles before the data to optimize performance on the ACP transfers.



# Chapter 10

## Monitoring, Trace, and Debug

This chapter describes the monitoring, trace, and debug features of the Cortex-R7 MPCore processor. It contains the following sections:

- *About monitoring, trace, and debug* on page 10-2.
- *Performance Monitoring Unit* on page 10-3.
- *Memory Reconstruction Port* on page 10-10.
- *Embedded Trace Macrocell* on page 10-11.
- *Debug* on page 10-12.

## 10.1 About monitoring, trace, and debug

The Cortex-R7 MPCore processor provides the following features for monitoring, trace, and debug

- [Performance Monitoring Unit](#) on page 10-3.
- [Memory Reconstruction Port](#) on page 10-10.
- [Embedded Trace Macrocell](#) on page 10-11.
- [Debug](#) on page 10-12.

## 10.2 Performance Monitoring Unit

The Cortex-R7 MPCore processor PMU provides eight counters to gather statistics on the operation of the processor and memory system. Each counter can count any of the 64 events available in each processor.

The PMU counters, and their associated control registers, are accessible from the internal CP15 interface and from the Debug APB interface. [Table 10-1](#) shows the mappings of the PMU registers.

**Table 10-1 Performance monitoring instructions and Debug APB mapping**

Debug APB interface mapping	CP15 instruction	Access	Reset	Name
0x000	0, c9, c13, 2	RW	-	PMXEVCNTR0
0x004	0, c9, c13, 2	RW	-	PMXEVCNTR1
0x008	0, c9, c13, 2	RW	-	PMXEVCNTR2
0x00C	0, c9, c13, 2	RW	-	PMXEVCNTR3
0x010	0, c9, c13, 2	RW	-	PMXEVCNTR4
0x014	0, c9, c13, 2	RW	-	PMXEVCNTR5
0x018	0, c9, c13, 2	RW	-	PMXEVCNTR6
0x01C	0, c9, c13, 2	RW	-	PMXEVCNTR7
0x07C	0, c9, c13, 0	RW	-	PMCCNTR
0x400	0, c9, c13, 1	RW	-	PMXEVTYPERO
0x404	0, c9, c13, 1	RW	-	PMXEVTYPER1
0x408	0, c9, c13, 1	RW	-	PMXEVTYPER2
0x40C	0, c9, c13, 1	RW	-	PMXEVTYPER3
0x410	0, c9, c13, 1	RW	-	PMXEVTYPER4
0x414	0, c9, c13, 1	RW	-	PMXEVTYPER5
0x418	0, c9, c13, 1	RW	-	PMXEVTYPER6
0x41C	0, c9, c13, 1	RW	-	PMXEVTYPER7
0xC00	0, c9, c12, 1	RW	-	PMCNTENSET
0xC20	0, c9, c12, 2	RW	-	PMCNTENCLR
0xC40	0, c9, c14, 1	RW	-	PMINTENSET
0xC60	0, c9, c14, 2	RW	-	PMINTENCLR
0xC80	0, c9, c12, 3	RW	-	PMOVSr
0xCA0	0, c9, c12, 4	WO	-	PMSWINC
0xE04	0, c9, c12, 0	RW	0x41174000	PMCR
0xE08	0, c9, c14, 0	RW <sup>a</sup>	0x00000000	PMUSERENR
-	0, c9, c12, 5	RW	-	PMSELR

a. Read only in user mode.

## 10.2.1 PMU management registers

The PMU management registers define the standardized set of registers that is implemented by all CoreSight components.

Table 10-2 shows the contents of the PMU management registers for the Cortex-R7 MPCore debug unit.

**Table 10-2 PMU management registers**

Offset	Register number	Access	Mnemonic	Description
0xD00-0xDFC	832-895	RO	-	<i>Processor ID Registers.</i>
0xE00-0xEF0	854-956	-	-	RAZ.
0xF04-0xF9C	961-999	RAZ	-	Reserved for Management Register expansion.
0xFA0	1000	RW	CLAIMSET	-
0xFA4	1001	RW	CLAIMCLR	-
0xFA8-0xFBC	1002-1003	-	-	RAZ.
0xFB0	1004	WO	LOCKACCESS	-
0xFB4		RO	LOCKSTATUS	-
0xFB8		RO	AUTHSTATUS	-
0xFBC-0xFC4	1007-1009	-	-	RAZ.
0xFC8	1010	RO	DEVID	Device Identifier.
0xFCC	1011	RO	DEVTYPE	-
0xFD0-0xFFC	1012-1023	R	-	<i>CoreSight Identification Registers on page 10-5.</i>

### Processor ID Registers

The Processor ID Registers are read-only registers that return the same values as the corresponding CP15 ID Code Register and Feature ID Register.

Table 10-3 shows the offset value, register number, mnemonic, and description that are associated with each Processor ID Register.

**Table 10-3 Processor Identifier Registers**

Offset (hex)	Register number	Mnemonic	Access	Register value	Description
0xD00	832	CPUID	RO	0x80000n0m <sup>a</sup>	ID Code Register
0xD04	833	CTR	RO	0x8333C003	Cache Type Register
0xD08	834	TCMTR	RO	0x80010001 <sup>b</sup> 0x00000000 <sup>c</sup>	TCM Type Register
0xD0C-0xD1C	835-839	-	-	-	Reserved
0xD20	840	ID_PFR0	RO	0x00000131	Processor Feature Register 0
0xD24	841	ID_PFR1	RO	0x00000001	Processor Feature Register 1

Table 10-3 Processor Identifier Registers (continued)

Offset (hex)	Register number	Mnemonic	Access	Register value	Description
0xD28	842	ID_DFR0	RO	0x00010404	Debug Feature Register 0
0xD2C	843	ID_AFR0	RAZ	-	Auxiliary Feature Register 0
0xD30	844	ID_MMFR0	RO	0x00110130	Memory Model Feature Register 0
0xD34	845	ID_MMFR1	RO	0x00000000	Memory Model Feature Register 1
0xD38	846	ID_MMFR2	RO	0x01200000	Memory Model Feature Register 2
0xD3C	847	ID_MMFR3	RO	0x00002111	Memory Model Feature Register 3
0xD40	848	ID_ISAR0	RO	0x02101111	Instruction Set Attribute Register 0
0xD44	849	ID_ISAR1	RO	0x13112111	Instruction Set Attribute Register 1
0xD48	850	ID_ISAR2	RO	0x21232141	Instruction Set Attribute Register 2
0xD4C	851	ID_ISAR3	RO	0x01112131	Instruction Set Attribute Register 3
0xD50	852	ID_ISAR4	RO	0x00010142	Instruction Set Attribute Register 4
0xD54	853	ID_ISAR5	RAZ	-	Instruction Set Attribute Register 5

- a.  $n = \text{CLUSTERID}$  input  $m$  = processor number (0x0 for processor 0, 0x1 for processor 1).
- b. If TCMs are present.
- c. If TCMs are not present.

### CoreSight Identification Registers

The Identification Registers are read-only registers that consist of the Peripheral Identification Registers and the Component Identification Registers. The Peripheral Identification Registers provide standard information required by all CoreSight components. Only bits[7:0] of each register are used.

The Component Identification Registers identify the processor as a CoreSight component. Only bits[7:0] of each register are used, the remaining bits Read-As-Zero. The values in these registers are fixed.

Table 10-4 shows the offset value, register number, value, and description that are associated with each Peripheral Identification Register.

Table 10-4 Peripheral Identification Registers

Offset (hex)	Register number	Value	Description
0xFD0	1012	0x04	Peripheral Identification Register 4
0xFD4	1013	-	Reserved
0xFD8	1014	-	Reserved
0xFDC	1015	-	Reserved
0xFE0	1016	0xB7	Peripheral Identification Register 0

**Table 10-4 Peripheral Identification Registers (continued)**

Offset (hex)	Register number	Value	Description
0xFE4	1017	0xB9	Peripheral Identification Register 1
0xFE8	1018	0xB <sup>a</sup>	Peripheral Identification Register 2
0xFEC	1019	0x00	Peripheral Identification Register 3

a. *r* represents the variant. For *r0p0*, this is 0.

Table 10-5 shows the offset value, register number, and value that are associated with each Component Identification Register.

**Table 10-5 Component Identification Registers**

Offset (hex)	Register number	Value	Description
0xFF0	1020	0x0D	Component Identification Register 0
0xFF4	1021	0x90	Component Identification Register 1
0xFF8	1022	0x05	Component Identification Register 2
0xFFC	1023	0xB1	Component Identification Register 3

## PMU APB interface

Table 10-6 shows the PMU register names and corresponding addresses on the APB interface.

**Table 10-6 PMU register names and APB addresses**

PMU register name	Debug APB address
PMU event counter 0	0x000
PMU event counter 1	0x004
PMU event counter 2	0x008
PMU event counter 3	0x00C
PMU event counter 4	0x010
PMU event counter 5	0x014
PMU event counter 6	0x018
PMU event counter 7	0x01C
PMU cycle counter	0x07C
PMU event type 0	0x400
PMU event type 1	0x404
PMU event type 2	0x408
PMU event type 3	0x40C
PMU event type 4	0x410

Table 10-6 PMU register names and APB addresses (continued)

PMU register name	Debug APB address
PMU event type 5	0x414
PMU event type 6	0x418
PMU event type 7	0x41C
PMU count enable set	0xC00
PMU count enable clear	0xC20
PMU interrupt enable set	0xC40
PMU interrupt enable clear	0xC60
PMU overflow flag status	0xC80
PMU software increment	0xCA0
PMU control	0xE04
PMU user enable	0xE08

## 10.2.2 Performance monitoring events

The Cortex-R7 MPCore processor implements the architectural events described in the *ARM Architecture Reference Manual*. For events and the corresponding **PMUEVENT** signals, see [Performance monitoring signals on page A-23](#).

The PMU provides an additional set of Cortex-R7 specific events.

The ETM accepts 64 events:

- Four events from the CTI.
- Four events are spare pins tied off at the ETM input level.
- 56 events from the processor, and visible externally through the **PMUEVENT<sub>n</sub>** bus, where *n* is 0 or 1.

The ETM can export two signals that are connected to the processor PMU. These signals are **ETMEXTOUT[1]** and **ETMEXTOUT[2]** events, and are described in [Table 10-7](#). See the *ARM® CoreSight™ ETM-R7 Technical Reference Manual* for more information about the **EXTOUT** signals from the ETM.

[Table 10-7](#) shows the Cortex-R7 MPCore processor events, with their associated event number, and position in the **PMUEVENT** bus.

Table 10-7 Cortex-R7 MPCore processor events

Event	Description	Position in PMUEVENT bus
Common events		
0x00	Software increment	[0]
0x01	Instruction cache miss	[1]
0x03	Data cache miss	[2]

Table 10-7 Cortex-R7 MPCore processor events (continued)

Event	Description	Position in PMUEVENT bus
0x04	Data cache access	[3]
0x06	Data read	[4]
0x07	Data write	[5]
0x08	Instruction architecturally executed	[11:6]
0x09	Exception taken	[12]
0x0A	Exception returns	[13]
0x0B	Write context ID	[14]
0x0C	Software change of PC	[15]
0x0D	Immediate branch	[16]
0x0E	Procedure return, other than exception return	[17]
0x0F	Unaligned	[18]
0x10	Branch mispredicted or not predicted	[19]
0x11	Cycle count	Not applicable
0x12	Predictable branches	[20]
0x14	Instruction cache access	[21]
ETM events		
0x40	<b>ETMEXTOUT[1]</b>	Not applicable
0x41	<b>ETMEXTOUT[2]</b>	Not applicable
Determinism events		
0x50	Number of cycles IRQs are interrupted	[22]
0x51	Number of cycles FIQs are interrupted	[23]
ECC events		
0x60	Detected ECC errors on any RAM	Not exported
0x61	Parity error on PRED	[24]
0x62	Parity error on BTAC	[25]
0x63	Detected ECC errors on ITCM	[26]
0x64	Detected ECC errors on DTCM	[27]
0x65	Detected ECC errors on instruction cache	[28]
0x66	Detected ECC errors on data cache	[29]
0x67	Correctable ECC errors on any bus	Not exported
0x68	Correctable ECC errors on slave bus, data write channel	[30]
0x69	Correctable ECC errors on peripheral master bus, data read channel	[31]



**Table 10-7 Cortex-R7 MPCore processor events (continued)**

Event	Description	Position in PMUEVENT bus
0x6A	Correctable ECC errors on master 0 bus, data read channel	[32]
0x6B	Correctable ECC errors on master 1 bus, data read channel	[33]
0x6C	Detected ECC errors on SCU RAM	[34]
Software events		
0x80	STREX passed	[35]
0x81	STREX failed	[36]
0x82	Literal pool in TCM region	[37]
Microarchitecture events		
0x90	DMB stall	[38]
0x91	ITCM access	[39]
0x92	DTCM access	[40]
0x93	Data eviction	[41]
0x94	SCU coherency operation (CCB request)	[42]
0x95	Instruction cache dependent stall	[43]
0x96 <sup>a</sup>	Data cache dependent stall <sup>b</sup>	[44]
0x97 <sup>a</sup>	Non-cacheable no peripheral dependent stall <sup>c</sup>	[45]
0x98 <sup>a</sup>	Non-cacheable peripheral dependent stall <sup>d</sup>	[46]
0x99 <sup>a</sup>	Data cache high priority dependent stall <sup>e</sup>	[47]
Reserved		
-	Reserved, tied LOW	[55:48]

- a. This counter is mostly used when QoS is enabled. The core issue stage is stalled and it contains at least one instruction that it cannot dispatch.
- b. The counter counts stalls on AXI M0 with low-priority Cacheable traffic.
- c. The counter counts stalls on AXI M0 or M1 with low-priority NC/SO/Dev.
- d. The counts stall on AXI MP with high-priority SO/DEV.
- e. The counts stall on AXI M1 with high-priority when a local SRAM is in use.

#### Note

You can choose whether these events are enabled or not, and exported or not, using the PMCR Register. See the *ARM® Architecture Reference Manual ARMv7-A and ARMv7-R edition*. For a fault-tolerant system, ECC events must be exported for more visibility. To achieve this, there are some specific ECC notification signals. See [Chapter 7 Fault Detection](#) for more information.

## 10.3 Memory Reconstruction Port

The MRP is an optional feature on the Cortex-R7 MPCore processor. All write accesses, regardless of memory attributes, such as Strongly-Ordered, Device, Non-cacheable, and Cacheable, are exported from the processor through this port so that an image of the memory can be reconstructed. This port is intended for memory reconstruction only.

The MRP can be enabled using the [Auxiliary Control Register on page 4-25](#).

The MRP has the following restrictions:

- Exclusive write accesses are reported on this interface only if they are successful. Failed exclusive write accesses never appear on this interface because they are never executed.
- Because TCM accesses are not mapped externally, they are not reported on this interface.
- When SWP accesses are atomic, the write access part of the SWP is exported on the interface when completed.
- One interface is provided per processor in a multiprocessor configuration. The implementer is responsible for synchronizing the different paths, if required, to be able to reconstruct an image of the memory.
- Write accesses are not in program order, apart from any architectural constraints on the memory attributes. If you require the accesses to be visible in order on the MRP:
  - Use Device or Strongly Ordered memory attributes.
  - Execute a *Data Synchronization Barrier* (DSB).

See [Memory reconstruction port signals on page A-39](#) for a complete list of the MRP interface signals.

The ready signal of the SoC slave is used to drive the ready signal of the store buffer towards the LSU. As a result, asserting this signal LOW directly impacts the performance of the processor, and writes are kept in the LSU until the SoC can execute the incoming writes. You can use a FIFO to provide limited and reasonable back-pressure on the ready signal.

---

### **Note**

---

There is no response channel on the MRP. Any possible dec/slave error is reported by the normal AXI channel.

---

## 10.4 Embedded Trace Macrocell

The optional ETM is compliant with the ETMv4 architecture. It provides full address and data trace, and enables real-time code tracing of the processor in an embedded system.

There is one ETM per processor, set up as a build option. In a twin-processor configuration, the processors can share a single ETM.

The ETM is enabled through the APB debug interface. You can disable the ETM, and power it off for power saving.

See the *ARM® CoreSight™ ETM-R7 Technical Reference Manual* for more information.

## 10.5 Debug

The Cortex-R7 MPCore processor implements the ARMv7 debug architecture and the set of debug events described in the *ARM® Architecture Reference Manual ARMv7-A and ARMv7-R edition*.

The following sections describe debug:

- [Debug events](#).
- [Debug registers](#).
- [External debug interface on page 10-19](#).
- [Trigger inputs and outputs on page 10-22](#).

### 10.5.1 Debug events

Depending on the programming of the debug control registers, debug events can:

- Generate debug exceptions, that is, software monitor debug.
- Make the processor enter debug state, that is, hardware halting debug.

The Cortex-R7 MPCore processor can handle the following debug events:

**Breakpoints** There are six breakpoints, two with Context ID comparison capability, BRP4 and BRP5.

**Watchpoints** There are four watchpoints. A watchpoint event is always synchronous. It has the same behavior as a synchronous data abort.

If a synchronous abort occurs on a watchpointed access, the synchronous abort takes priority over the watchpoint.

If the abort is asynchronous and cannot be associated with the access, the exception that is taken is UNPREDICTABLE.

Cache maintenance operations do not generate watchpoint events.

#### Other debug events

The Cortex-R7 MPCore processor implements:

- Vector catch.
- BKPT instruction.
- External debug request.
- Halt request.

#### ————— Note —————

The Cortex-R7 MPCore processor does not implement the OS catch debug event.

See the *ARM® Architecture Reference Manual ARMv7-A and ARMv7-R edition* for more information.

### 10.5.2 Debug registers

The following sections describe the debug registers:

- [Register interfaces on page 10-13](#).
- [Debug register mapping on page 10-13](#).
- [Debug register descriptions on page 10-15](#).
- [Effects of resets on debug registers on page 10-16](#).
- [Debug management registers on page 10-16](#).

## Register interfaces

The Cortex-R7 MPCore processor implements Baseline CP14, Extended CP14, and memory-mapped interfaces. You can access the debug registers as follows:

- Through the cp14 interface. The debug registers are mapped to coprocessor instructions.
- Through the APB using the relevant offset.

## Debug register mapping

Table 10-8 shows the debug register mapping. All other registers are described in the *ARM Architecture Reference Manual*.

**Table 10-8 Debug register mapping**

Register number	APB offset	APB access	CP14 address	CP14 access	Register name	Description
0	0x000	RO	0, c0, c0, 0	RO	DBGDIDR <sup>a</sup>	.. <sup>b</sup>
128	No access	No access	0, c1, c0, 0	RO	DBGDRAR <sup>a</sup>	-
256	No access	No access	0, c2, c0, 0	RO	DBGDSAR <sup>a</sup>	-
1	No access	No access	0, c0, c1, 0	RO	DBGDSCRint <sup>ab</sup>	-
5	No access	No access	0, c0, c5, 0	RO	DBGDTRRXint <sup>a</sup>	-
	No access	No access		WO	DBGDTRTXint <sup>a</sup>	-
6	0x018	RW	0, c0, c6, 0	RW	DBGWFAR	Use of DBGWFAR is deprecated in the ARMv7 architecture, because watchpoints are synchronous
7	0x01C	RW	0, c0, c7, 0	RW	DBGVCR	-
8	-	-	-	-	Reserved	-
9	No access	No access	0, c0, c9, 0	RAZ/WI	DBGECR	Not implemented
10	No access	No access	0, c0, c10, 0	RAZ/WI	DBGDSCCR	Not implemented
11	No access	No access	0, c0, c11, 0	RAZ/WI	DBGDSMCR	Not implemented
12-31	-	-	-	-	Reserved	-
32	0x080	RW	0, c0, c0, 2	RW	DBGDTRRXext	-
33	0x084	WO	0, c0, c1, 2	WO	DBGITR	-
33	0x084	RO	0, c0, c1, 2	RO	DBGPCSR	-
34	0x088	RW	0, c0, c2, 2	RW	DBGDSCRext	-
35	0x08C	RW	0, c0, c3, 2	RW	DBGDTRTXext	-
36	0x090	WO	0, c0, c4, 2	WO	DBGDRCR	-
37-63	-	-	-	-	Reserved	-
64-69	0x100-0x114	RW	0, c0, c0-c5, 4	RW	DBGBVRn	Breakpoint Value Registers

Table 10-8 Debug register mapping (continued)

Register number	APB offset	APB access	CP14 address	CP14 access	Register name	Description
70-79	-	-	-	-	Reserved	-
80-85	0x140-0x154	RW	0, c0, c0-c5, 5	RW	DBGBCRn	Breakpoint Control Registers
86-95	-	-	-	-	Reserved	-
96-99	0x180-0x18C	RW	0, c0, c0-c3, 6	RW	DBGWVRn	Watchpoint Value Registers
100-111	-	-	-	-	Reserved	-
112-115	0x1C0-0x1CC	RW	0, c0, c0-c3, 7	RW	DBGWCRn	Watchpoint Control Registers
116-191	-	-	-	-	Reserved	-
192	0x300	RAZ/WI	0, c1, c0, 4	RAZ/WI	DBGOSLAR	Not implemented
193	0x304	RAZ/WI	0, c1, c1, 4	RAZ/WI	DBGOSLSR	Not implemented
194	0x308	RAZ/WI	0, c1, c2, 4	RAZ/WI	DBGOSSRR	Not implemented
195	-	-	-	-	Reserved	-
196	0x310	RW	0, c1, c4, 4	RW	DBGPRCR	-
197	0x314	RO	0, c1, c5, 4	RO	DBGPRSR	-
198-511	-	-	-	-	Reserved	-
512-575	0x800-0x8FC	-	-	-	-	PMU registers <sup>c</sup>
576-831	-	-	-	-	Reserved	-
832-895	0xD00-0xDFC	-	-	-	-	<i>Processor ID Registers on page 10-4</i>
896-927	-	-	-	-	Reserved	-
928-959	0xE80-0xEFC	RAZ/WI	No access	No access	-	-
960	0xF00	RAZ/WI	0, c7, c0, 4	RAZ/WI	DBGITCTRL	Integration Mode Control Register
961-999	0xF04-0xF9C	-	-	-	-	-
1000	0xFA0	RW	0, c7, c8, 6	RW	DBGCLAIMSET	Claim Tag Set Register
1001	0xFA4	RW	0, c7, c9, 6	RW	DBGCLAIMCLR	Claim Tag Clear Register
1002-1003	-	-	-	-	Reserved	-
1004	0xFB0	WO	No access	No access	DBGLAR	Lock Access Register
1005	0xFB4	RO	No access	No access	DBGLSR	Lock Status Register
1006	0xFB8	RO	0, c7, c14, 6	RO	DBGAUTHSTATUS	Authentication Status Register
1007-1008	-	-	-	-	Reserved	-
1009	0xFC4	RO	No access	No access	DBGDEVID1	-

Table 10-8 Debug register mapping (continued)

Register number	APB offset	APB access	CP14 address	CP14 access	Register name	Description
1010	0xFC8	RO	No access	No access	DBGDEVID	-
1011	0xFCC	RO	No access	No access	DBGDEVTYPE	Device Type Register
1012-1016	0xFD0-0xFEC	RO	No access	No access	PERIPHERALID	<i>CoreSight Identification Registers on page 10-18</i>
1017-1019	-	-	-	-	Reserved	-
1020-1023	0xFF0-0xFFC	RO	No access	No access	COMPONENTID	<i>CoreSight Identification Registers on page 10-18</i>

- a. Baseline CP14 interface. This register also has an external view through the memory-mapped interface and the CP14 interface.
- b. Accessible in user mode if bit [12] of the DBGSCR is clear. Also accessible in privileged modes.
- c. PMU registers are part of the CP15 interface. Reads from the extended CP14 interface return zero. See *Register summary on page 4-3*. See also *Performance Monitoring Unit on page 10-3*.

### Debug register descriptions

This section describes register features specific to the Cortex-R7 MPCore processor. See the *ARM Architecture Reference Manual* for information about other register features not described in this section.

#### Debug Status and Control Register, **DBGDSCR**

Behaves as described in the *ARM Architecture Reference Manual*, except for the following bits:

**PipeAdv, bit[25]** This bit is set each time a branch is resolved in the processor.

**HALTED, bit[0]** This bit is the only bit of the register that is not reset on debug logic reset. It is reset to 1'b0 on a processor logic reset. Its behavior is as described in the *ARM Architecture Reference Manual*.

#### Debug Run Control Register, **DBGDRCR**

Behaves as described in the *ARM Architecture Reference Manual*, except for the following bits:

**Cancel BIU Requests, bit[4]**  
Not implemented, RAZ/WI.

**Bits[3:0]** Implemented as described in the *ARM Architecture Reference Manual*.

#### Device Powerdown and Reset Control Register, **DBGPRCR**

Behaves as described in the *ARM Architecture Reference Manual*, except for the following bits:

**Bits[2:1]** Not implemented, RAZ/WI.

**DBGnoPWRDWN, bit[0]**  
Implemented as described in the *ARM Architecture Reference Manual* (RW).

**Device Powerdown and Reset Status Register, DBGPRSR**

Behaves as described in the *ARM Architecture Reference Manual*, except for the following bits:

**Sticky Reset Status, bit[3]**

Implemented as described in the *ARM Architecture Reference Manual*.

**Reset Status, bit[2]**

Implemented as described in the *ARM Architecture Reference Manual*.

**Sticky Powerdown Status, bit[1]**

Not implemented, RAZ/WI.

**Power-up Status, bit[0]**

Implemented, RAO.

**Breakpoint and Watchpoint Registers, DBGBVRn, DBGBCRn, DBGWVRn, and DBGWCRn**

Behave as described in the *ARM Architecture Reference Manual*, except for the following:

- Only BRP4 and BRP5 support context ID comparison.
- BVR0[1:0], BVR1[1:0], BVR2[1:0], and BVR3[1:0] are SBZP on writes and RAZ on reads because these registers do not support context ID comparisons.
- The context ID value for a BVR to match with is given by the contents of the CP15 Context ID Register.

**Effects of resets on debug registers****nDBGRESET**

**nDBGRESET** is the debug logic reset signal. This signal must be asserted during a powerup reset sequence.

On a debug reset:

- The debug state is unchanged. That is, DBGSCR.HALTED is unchanged.
- The processor removes the pending halting debug events DBGDRCR.HaltReq.

**Debug management registers**

The management registers define the standardized set of registers that is implemented by all CoreSight components. [Table 10-9](#) shows the contents of the debug management registers for the Cortex-R7 debug unit. On the Cortex-R7 MPCore processor, the debug management registers are memory-mapped.

**Table 10-9 Debug management registers**

APB offset	Register number	Access	Mnemonic	Description
0xD00-0xDFC	832-895	RO	-	<a href="#">Processor ID Registers on page 10-17</a>
0xE00-0xEF0	854-956	RAZ/WI	-	Not implemented
0xF00	960	RAZ/WI	ITCTRL	-
0xF04-0xF9C	961-999	RAZ/WI	-	Not implemented
0xFA0	1000	RW	CLAIMSET	-



Table 10-9 Debug management registers (continued)

APB offset	Register number	Access	Mnemonic	Description
0xFA4	1001	RW	CLAIMCLR	-
0xFA8-0xFBC	1002-1003	RAZ/WI	-	Not implemented
0xFB0	1004	WO	LOCKACCESS	-
0xFB4		RO	LOCKSTATUS	-
0xFB8		RO	AUTHSTATUS	-
0xFBC-0xFC4	1007-1009	RAZ/WI	-	Not implemented
0xFC8	1010	RO	DEVID	-
0xFCC	1011	RO	DEVTYPE	-
0xFD0-0xFFC	1012-1023	RO	-	<i>CoreSight Identification Registers on page 10-18</i>

**Processor ID Registers**

The Processor ID Registers are read-only registers that return the same values as the corresponding CP15 ID Code Register and Feature ID Register.

Table 10-10 shows the APB offset value, register number, mnemonic, and description that are associated with each Processor ID Register.

Table 10-10 Processor ID Registers

APB offset	Register number	Mnemonic	Access	Register value	Description
0xD00	832	MIDR	RO	<i>_a</i>	Main ID Register alias
0xD04	833	CTR	RO	0x8333C003	Cache Type Register
0xD08	834	TCMTR	RO	<i>_b</i>	TCM Type Register
0xD0C	835	MIDR	RO	<i>_a</i>	Main ID Register alias
0xD10	836	MPUIR	RO	<i>_c</i>	MPU Type Register
0xD14	837	MPIDR	RO	<i>_d</i>	Multiprocessor Affinity Register
0xD18	838	REVIDR	RO	0x0	Revision ID Register
0xD1C	839	MIDR	RO	<i>_a</i>	Main ID Register alias
0xD20	840	ID_PFR0	RO	0x00000131	Processor Feature Register 0
0xD24	841	ID_PFR1	RO	0x00000001	Processor Feature Register 1
0xD28	842	ID_DFR0	RO	0x00010404	Debug Feature Register 0
0xD2C	843	ID_AFR0	RAZ	-	Auxiliary Feature Register 0
0xD30	844	ID_MMFR0	RO	0x00110130	Memory Model Feature Register 0
0xD34	845	ID_MMFR1	RO	0x0	Memory Model Feature Register 1
0xD38	846	ID_MMFR2	RO	0x01200000	Memory Model Feature Register 2
0xD3C	847	ID_MMFR3	RO	0x00002111	Memory Model Feature Register 3

**Table 10-10 Processor ID Registers (continued)**

APB offset	Register number	Mnemonic	Access	Register value	Description
0xD40	848	ID_ISAR0	RO	0x02101111	Instruction Set Attribute Register 0
0xD44	849	ID_ISAR1	RO	0x13112111	Instruction Set Attribute Register 1
0xD48	850	ID_ISAR2	RO	0x21232141	Instruction Set Attribute Register 2
0xD4C	851	ID_ISAR3	RO	0x01112131	Instruction Set Attribute Register 3
0xD50	852	ID_ISAR4	RO	0x00010142	Instruction Set Attribute Register 4
0xD54	853	ID_ISAR5	RAZ	-	Instruction Set Attribute Register 5

- a. 0x41nFC17m:  
n = variant.  
m = revision.
- b. TCM present = 0x80010001.  
TCM not present = 0x0.
- c. MPU\_16 configuration = 0x00001000.  
MPU\_12 configuration = 0x0000C000.
- d. Dependent on cluster and processor IDs:  
With one processor = 0x80000000.  
With two processors = 0xC0000n0m.  
n = **CLUSTERID** input.  
m = processor number (0x0 for processor 0, 0x1 for processor 1).

### CoreSight Identification Registers

The Identification Registers are read-only registers that consist of the Peripheral Identification Registers and the Component Identification Registers. The Peripheral Identification Registers provide standard information required by all CoreSight components. Only bits[7:0] of each register are used.

The Component Identification Registers identify the processor as a CoreSight component. Only bits[7:0] of each register are used, the remaining bits Read-As-Zero. The values in these registers are fixed.

[Table 10-11](#) shows the APB offset value, register number, and description that are associated with each Peripheral Identification Register.

**Table 10-11 Peripheral Identification Registers for processor debug**

APB offset	Register number	Value	Description
0xFD0	1012	0x04	Peripheral Identification Register 4
0xFD4	1013	-	Reserved
0xFD8	1014	-	Reserved
0xFDC	1015	-	Reserved
0xFE0	1016	0x17	Peripheral Identification Register 0
0xFE4	1017	0xBC	Peripheral Identification Register 1
0xFE8	1018	0x0B	Peripheral Identification Register 2
0xFEC	1019	0x00	Peripheral Identification Register 3

Table 10-12 shows the APB offset value, register number, and value that are associated with each Component Identification Register.

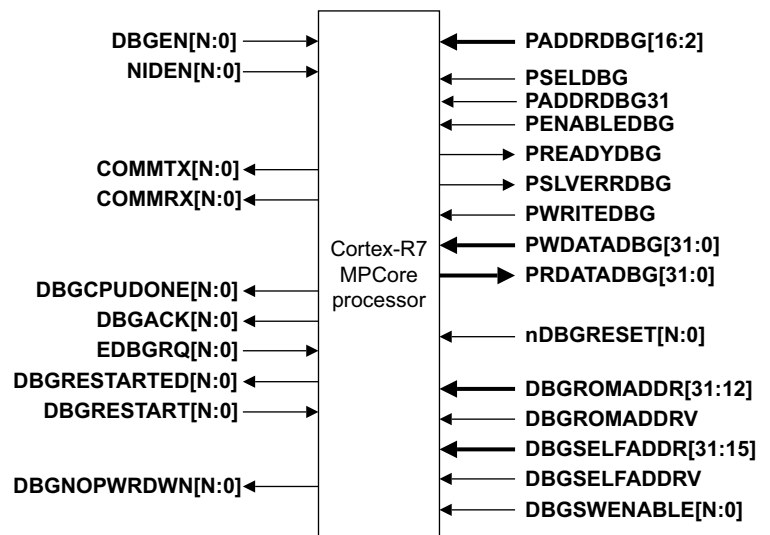
**Table 10-12 Component Identification Registers**

APB offset	Register number	Value	Description
0xFF0	1020	0x0D	Component Identification Register 0
0xFF4	1021	0x90	Component Identification Register 1
0xFF8	1022	0x05	Component Identification Register 2
0xFFC	1023	0xB1	Component Identification Register 3

### 10.5.3 External debug interface

See [External debug signals on page A-34](#) for a complete list of the external debug signals.

Figure 10-1 shows the external debug interface signals.



**Figure 10-1 External debug interface signals**

### Authentication signals

Table 10-13 shows a list of the valid combinations of authentication signals along with their associated debug permissions.

**Table 10-13 Authentication signal restrictions**

DBGEN	NIDEN	Invasive debug permitted	Non-invasive debug permitted
0	0	No	No
0	1	No	Yes
1	0	Yes	Yes
1	1	Yes	Yes

## Debug APB interface

The system can access memory-mapped debug registers through the Cortex-R7 MPCore APB slave port. This APB slave interface supports 32-bit wide data, stalls, and slave-generated aborts. [Table 10-14](#) shows the mapping of **PADDRDBG[16:2]**.

**Table 10-14 PADDRDBG[16:2] mapping**

Address map	Reset domain	Component
0x00000-0x00FFF	Integration	ROM Table.
0x01000-0x0FFFF	-	Reserved.
0x10000-0x10FFF	CPU0	Processor 0 debug. See <a href="#">Table 10-8 on page 10-13</a> for debug resource memory mapping.
0x11000-0x11FFF	CPU0	Processor 0 PMU. See <a href="#">Performance Monitoring Unit on page 10-3</a> for PMU resource mapping.
0x12000-0x12FFF	CPU1	Processor 1 Debug. See <a href="#">Table 10-8 on page 10-13</a> for debug resource memory mapping.
0x13000-0x13FFF	CPU1	Processor 1 PMU. See <a href="#">Performance Monitoring Unit on page 10-3</a> for PMU resource mapping.
0x14000-0x17FFF	-	Reserved.
0x18000-0x18FFF	Integration	Processor 0 CTL.
0x19000-0x19FFF	Integration	Processor 1 CTL.
0x20000-0x1BFFF	-	Reserved.
0x1C000-0x1CFFF	Integration	Processor 0 ETM.
0x1D000-0x1DFFF	Integration	Processor 1 ETM.
0x1E000-0x1FFFF	-	Reserved.

The **PADDRDBG31** signal indicates the source of the access to the processor.

## External debug request interface

The following sections describe the external debug request interface signals:

- [EDBGRQ](#).
- [DBGACK](#).
- [DBGCPUDONE on page 10-21](#).
- [COMMRX and COMMTX on page 10-21](#).
- [DBGROMADDR and DBGSELFADDR on page 10-21](#).

### EDBGRQ

This signal generates a halting debug event by requesting the processor to enter debug state. When this occurs, the DSCR[5:2] method of debug entry bits are set to 0b0100. When **EDBGRQ** is asserted, it must be held until **DBGACK** is asserted. Failure to do so causes UNPREDICTABLE behavior of the processor.

### DBGACK

The processor asserts **DBGACK** to indicate that the system has entered debug state. It serves as a handshake for the **EDBGRQ** signal. The **DBGACK** signal is also driven HIGH when the debugger sets the DSCR[10] DbgAck bit to 1.

### DBGCPUDONE

**DBGCPUDONE** is asserted when the processor has completed a *Data Synchronization Barrier* (DSB) as part of the entry procedure to debug state.

The processor asserts **DBGCPUDONE** only after it has completed all Non-debug state memory accesses. Therefore the system can use **DBGCPUDONE** as an indicator that all memory accesses issued by the processor result from operations performed by a debugger.

Figure 10-2 shows the Cortex-R7 MPCore processor connections specific to debug request and restart and the CoreSight inputs and outputs.

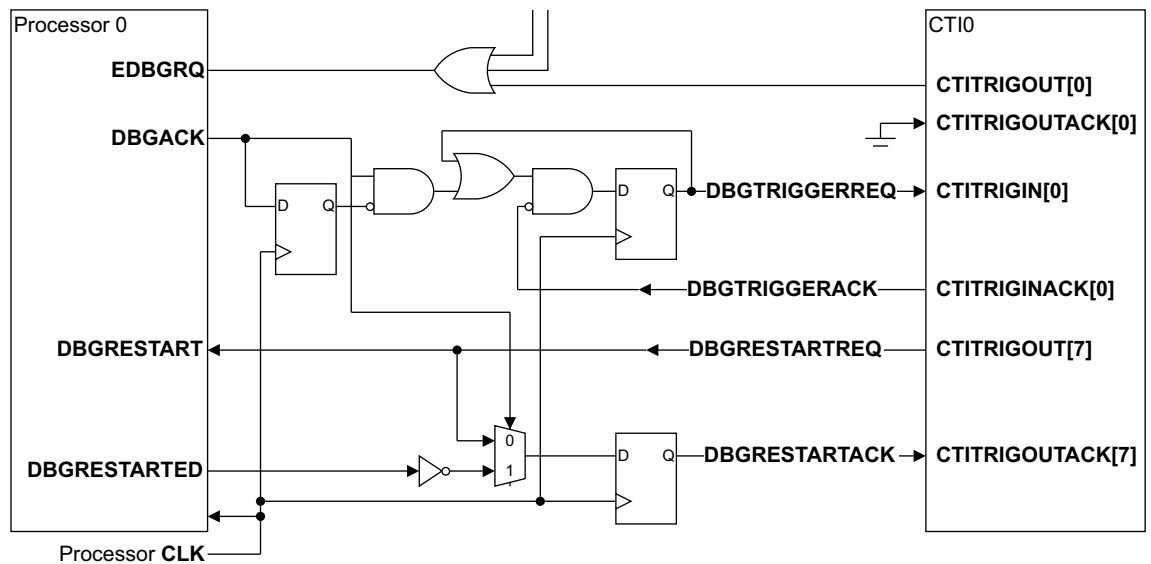


Figure 10-2 Debug request restart-specific connections

### COMMRX and COMMTX

The **COMMRX** and **COMMTX** output signals enable interrupt-driven communications over the DTR. By connecting these signals to an interrupt controller, software using the debug communications channel can be interrupted whenever there is new data on the channel or when the channel is clear for transmission:

- **COMMRX** is asserted when the CP14 DTR has data for the processor to read, and is deasserted when the processor reads the data. Its value is equal to the DBGDSCR[30] DTRRX full flag.
- **COMMTX** is asserted when the CP14 DTR is ready for write data, and is deasserted when the processor writes the data. Its value is equal to the inverse of the DBGDSCR[29] DTRTX full flag.

### DBGROMADDR and DBGSELFADDR

Cortex-R7 processors have a memory-mapped debug interface, and can access the debug and PMU registers by executing load and store instructions going through the AXI3 bus:

- **DBGROMADDR** gives the base address for the ROM table that locates the physical addresses of the debug components.
- **DBGSELFADDR** gives the offset from the ROM table to the physical addresses of the registers in the processor itself.

## 10.5.4 Trigger inputs and outputs

This section describes the trigger inputs and outputs that are available to the CTI.

Table 10-15 shows the CTI inputs.

**Table 10-15 Trigger inputs**

CTI input	Name	Description
0	<b>DBGTRIGGER</b> , pulsed	Pulsed on entry to debug state
1	<b>PMUIRQ</b>	PMU generated interrupt
2	<b>EXTOUT[0]</b>	ETM external output
3	<b>EXTOUT[1]</b>	ETM external output
4	<b>EXTOUT[2]</b>	ETM external output
5	<b>EXTOUT[3]</b>	ETM external output
6	<b>COMMTX</b>	Debug communication transmit channel is empty
7	<b>COMMRX</b>	Debug communication receive channel is full

Table 10-16 shows the CTI outputs.

**Table 10-16 Trigger outputs**

CTI output	Name	Description
0	<b>EDBGRQ</b>	Causes the processor to enter debug state
1	<b>EXTIN[0]</b>	ETM external input - ETM event
2	<b>EXTIN[1]</b>	ETM external input - ETM event
3	<b>EXTIN[2]</b>	ETM external input - ETM event
4	<b>EXTIN[3]</b>	ETM external input - ETM event
5	-	-
6	<b>nCTIIRQ</b>	CTI interrupt
7	<b>DBGRESTART</b>	Causes the processor to exit debug state

# Chapter 11

## Level Two Interface

This chapter describes the L2 memory interface. It contains the following sections:

- *About the L2 interface* on page 11-2.
- *Optimized accesses to the L2 memory interface* on page 11-8.
- *Accessing RAMs using the AXI3 interface* on page 11-9.
- *STRT instructions* on page 11-10.
- *Event communication with an external agent using WFE/SEV* on page 11-11.
- *Accelerator Coherency Port interface* on page 11-12.

## 11.1 About the L2 interface

The Cortex-R7 L2 interface consists of one or two 64-bit wide AXI3 bus masters:

- AXI3 M0 is always present.
- AXI3 M1 has support for address filtering.

Table 11-1 shows the AXI3 master 0 and master 1 interface attributes.

**Table 11-1 AXI3 master 0 and master 1 interface attributes**

Attribute	Format
Write issuing capability	29 for an implementation with one processor, including: <ul style="list-style-type: none"> <li>• 15 ACP writes.</li> <li>• Eight noncacheable writes.</li> <li>• Four evictions.</li> <li>• Two DDI evictions.</li> </ul> 41 for an implementation with two processors, including: <ul style="list-style-type: none"> <li>• 15 ACP writes.</li> <li>• For each processor:               <ul style="list-style-type: none"> <li>— Eight noncacheable writes.</li> <li>— Four evictions.</li> </ul> </li> <li>• Two DDI evictions.</li> </ul>
Read issuing capability	31 for an implementation with one processor, including: <ul style="list-style-type: none"> <li>• 15 ACP reads.</li> <li>• Four data side linefill reads.</li> <li>• Eight instruction side linefill reads.</li> <li>• Four noncacheable reads.</li> </ul> 47 for an implementation with two processors, including: <ul style="list-style-type: none"> <li>• 15 ACP reads.</li> <li>• For each processor:               <ul style="list-style-type: none"> <li>— Four data side linefill reads.</li> <li>— Eight instruction side linefill reads.</li> <li>— Four noncacheable reads.</li> </ul> </li> </ul>
Combined issuing capability	88
Write interleave capability	1

### Note

The numbers in Table 11-1 are the theoretical maximums for the Cortex-R7 MPCore processor. A typical system is unlikely to reach these numbers. ARM recommends that you perform profiling to tailor your system resources appropriately for optimum performance.

The AXI3 protocol and meaning of each AXI3 signal are not described in this document. For more information see the *AMBA AXI Protocol Specification*.

The following sections describe the Cortex-R7 L2 interface:

- [Supported AXI3 transfers on page 11-3.](#)
- [AXI3 USER bits on page 11-3.](#)



### 11.1.1 Supported AXI3 transfers

The Cortex-R7 MPCore processor master ports can generate all possible AXI3 transactions from ACP traffic.

Transactions from the individual processors use only the following subset of possible AXI3 transactions:

- For cacheable transactions:
  - WRAP4 64-bit for read transfers (linefills).
  - INCR4 64-bit for write transfers (evictions).
- For non-cacheable transactions:
  - WRAP4 64-bit for NC reads from instruction cache.
  - INCR N (N:1- 4) 64-bit read transfers.
  - INCR 1 for 64-bit write transfers.
  - INCR N (N:1-8) 32-bit read transfers.
  - INCR N (N:1-2) for 32-bit write transfers.
  - INCR 1 for 8-bit and 16-bit read/write transfers.
  - INCR 1 for 8-bit, 16-bit, 32-bit, and 64-bit exclusive read/write transfers.
  - INCR 1 for 8-bit and 32-bit read/write (locked) for swap.

The following points apply to AXI3 transactions:

- WRAP bursts are only read transfers, 64-bit, 4 transfers.
- INCR 1 can be any size for read or write.
- INCR bursts (more than one transfer) are only 32-bit or 64-bit.
- No transaction is marked as FIXED.
- Write transfers with all byte strobes low can occur.

### 11.1.2 AXI3 USER bits

The following sections describe the AXI3 USER bits encodings:

- *Read address channel of AXI master 0, ARUSERM0[8:0] on page 11-4.*
- *Read address channel of AXI master 1, ARUSERM1[8:0] on page 11-4.*
- *Peripheral read bus, ARUSERMP[8:0] on page 11-5.*
- *Write address channel of AXI master 0, AWUSERM0[10:0] on page 11-5.*
- *Write address channel of AXI master 1, AWUSERM1[10:0] on page 11-6.*
- *Peripheral write bus, AWUSERMP[10:0] on page 11-6.*
- *Write data channel of AXI master 0, WUSERM0[1:0] on page 11-7.*
- *Write data channel of AXI master 1, WUSERM1[1:0] on page 11-7.*
- *Peripheral write data bus, WUSERMP[1:0] on page 11-7.*

**Read address channel of AXI master 0, ARUSERM0[8:0]**

Table 11-2 shows the bit encodings for **ARUSERM0[8:0]**

**Table 11-2 ARUSERM0[8:0] encodings**

Bits	Name	Description
[8:7]	Transaction type	0b00 Processor 0 transaction.
		0b10 Processor 1 transaction.
		0b01 ACP transaction.
		0b11 Reserved.
[6]	Speculative linefill hint	Speculative linefill, used with L2C-310 Cache Controller.
[5]	Reserved	0b0
[4:1]	Inner attributes	0b0000 Strongly Ordered.
		0b0001 Device.
		0b0011 Normal Memory Non-Cacheable.
		0b0110 Reserved. <sup>a</sup>
		0b0111 Write Back no Write Allocate.
[0]	Shareable bit	0b0 Non-shareable.
		0b1 Shareable.

a. If Write Through is used in the MPU, it behaves as normal memory, non-cacheable, and its value is 0b0110.

**Read address channel of AXI master 1, ARUSERM1[8:0]**

Table 11-3 shows the bit encodings for **ARUSERM1[8:0]**.

**Table 11-3 ARUSERM1[8:0] encodings**

Bits	Name	Description
[8:7]	Transaction type	0b00 Processor 0 transaction.
		0b10 Processor 1 transaction.
		0b01 ACP transaction.
		0b11 Reserved.
[6]	Speculative linefill hint	Speculative linefill, used with L2C-310 Cache Controller.
[5]	Reserved	0b0
[4:1]	Inner attributes	0b0000 Strongly Ordered.
		0b0001 Device.
		0b0011 Normal Memory Non-Cacheable.
		0b0110 Reserved. <sup>a</sup>
		0b0111 Write Back no Write Allocate.
[0]	Shareable bit	0b0 Non-shareable.
		0b1 Shareable.

a. If Write Through is used in the MPU, it behaves as normal memory, non-cacheable, and its value is 0b0110.

**Peripheral read bus, ARUSERMP[8:0]**

Table 11-4 shows the bit encodings for **ARUSERMP[8:0]**.

**Table 11-4 ARUSERMP[8:0] encodings**

Bits	Name	Description
[8:7]	Transaction type	0b00 Processor 0 transaction.
		0b10 Processor 1 transaction.
		0b01 ACP transaction.
		0b11 Reserved.
[6:5]	Reserved	0b0
[4:1]	Inner attributes	0b0000 Strongly Ordered.
		0b0001 Device.
		0b0011 Normal Memory Non-Cacheable.
		0b0110 Write Through.
		0b0111 Write Back no Write Allocate.
		0b1111 Write Back Write Allocate.
[0]	Shareable bit	0b0 Non-shareable.
		0b1 Shareable.

**Write address channel of AXI master 0, AWUSERM0[10:0]**

Table 11-5 shows the bit encodings for **AWUSERM0[10:0]**.

**Table 11-5 AWUSERM0[10:0] encodings**

Bits	Name	Description
[10:9]	Transaction type	0b00 Processor 0 transaction.
		0b10 Processor 1 transaction.
		0b01 ACP transaction.
		0b11 Reserved.
[8]	Early BRESP Enable bit	Indicates that the L2 slave can send an early BRESP answer to the write request. See <a href="#">Early BRESP</a> on page 11-8.
[7:5]	Reserved	RAZ
[4:1]	Inner attributes	0b0000 Strongly Ordered.
		0b0001 Device.
		0b0011 Normal Memory Non-Cacheable.
		0b0110 Reserved. <sup>a</sup>
		0b0111 Write Back no Write Allocate.
		0b1111 Write Back Write Allocate.
[0]	Shareable bit	0b0 Non-shareable.
		0b1 Shareable.

a. If Write Through is used in the MPU, it behaves as normal memory, non-cacheable, and its value is 0b0110.

**Write address channel of AXI master 1, AWUSERM1[10:0]**

Table 11-6 shows the bit encodings for **AWUSERM1[10:0]**.

**Table 11-6 AWUSERM1[10:0] encodings**

Bits	Name	Description
[10:9]	Transaction type	0b00 Processor 0 transaction.
		0b10 Processor 1 transaction.
		0b01 ACP transaction.
		0b11 Reserved.
[8]	Early BRESP Enable bit	Indicates that the L2 slave can send an early BRESP answer to the write request. See <a href="#">Early BRESP on page 11-8</a> .
[7:5]	Reserved	RAZ
[4:1]	Inner attributes	0b0000 Strongly Ordered.
		0b0001 Device.
		0b0011 Normal Memory Non-Cacheable.
		0b0110 Reserved. <sup>a</sup>
		0b0111 Write Back no Write Allocate.
		0b1111 Write Back Write Allocate.
[0]	Shareable bit	0b0 Non-shareable.
		0b1 Shareable.

a. If Write Through is used in the MPU, it behaves as normal memory, non-cacheable, and its value is 0b0110.

**Peripheral write bus, AWUSERMP[10:0]**

Table 11-7 shows the bit encodings for **AWUSERMP[10:0]**.

**Table 11-7 AWUSERMP[10:0] encodings**

Bits	Name	Description
[10:9]	Transaction type	0b00 Processor 0 transaction.
		0b10 Processor 1 transaction.
		0b01 ACP transaction.
		0b11 Reserved.
[8:5]	Reserved	RAZ
[4:1]	Inner attributes	0b0000 Strongly Ordered.
		0b0001 Device.
		0b0011 Normal Memory Non-Cacheable.
		0b0110 Reserved. <sup>a</sup>
		0b0111 Write Back no Write Allocate.
		0b1111 Write Back Write Allocate.
[0]	Shareable bit	0b0 Non-shareable.
		0b1 Shareable.

a. If Write Through is used in the MPU, it behaves as normal memory, non-cacheable, and its value is 0b0110.

**Write data channel of AXI master 0, WUSERM0[1:0]**

Table 11-8 shows the bit encodings for **WUSERM0[1:0]**.

**Table 11-8 WUSERM0[1:0] encodings**

Bits	Name	Description
[1:0]	Transaction type	0b00 Processor 0 transaction.
		0b10 Processor 1 transaction.
		0b01 ACP transaction.
		0b11 Reserved.

**Write data channel of AXI master 1, WUSERM1[1:0]**

Table 11-9 shows the bit encodings for **WUSERM1[1:0]**.

**Table 11-9 WUSERM1[1:0] encodings**

Bits	Name	Description
[1:0]	Transaction type	0b00 Processor 0 transaction.
		0b10 Processor 1 transaction.
		0b01 ACP transaction.
		0b11 Reserved.

**Peripheral write data bus, WUSERMP[1:0]**

Table 11-10 shows the bit encodings for **WUSERMP[1:0]**.

**Table 11-10 WUSERMP[1:0] encodings**

Bits	Name	Description
[1:0]	Transaction type	0b00 Processor 0 transaction.
		0b10 Processor 1 transaction.
		0b01 ACP transaction.
		0b11 Reserved.

## 11.2 Optimized accesses to the L2 memory interface

This section describes optimized accesses to the L2 memory interface. These optimized accesses can generate non-AXI3 compliant requests on the AXI master ports. These non-AXI3 compliant requests must be generated only when the slaves connected on the AXI master ports can support them. The L2C-310 Cache Controller supports these types of requests. The following subsections describe the requests:

- [Early BRESP](#).
- [SCU speculative coherent requests](#).

### 11.2.1 Early BRESP

According to the AXI3 specification, **BRESP** answers on response channels must be returned to the master only when the last data has been sent by the master. Cortex-R7 MPCore processors can also deal with **BRESP** answers returned as soon as address has been accepted by the slave, regardless of whether data is sent or not. This enables the processor to provide a higher bandwidth for writes if the slave can support the Early BRESP feature. Cortex-R7 MPCore processors set the **AWUSER[8]** bit to indicate to the slave that it can accept an early **BRESP** answer for this access. This feature can optimize the performance of the processor, but the Early BRESP feature generates non-AXI3 compliant requests. When a slave receives a write request with **AWUSER[8]** set, it can either give the BRESP answer after the last data is received, AXI3 compliant, or in advance, non-AXI3 compliant. The L2C-310 Cache Controller supports this non-AXI3 compliant feature.

This feature is enabled by default. The Cortex-R7 MPCore processor does not require any programming to enable this feature.

---

#### Note

To support this optimization, you must program the L2C-310 Cache Controller. See the *CoreLink Level 2 Cache Controller (L2C-310) Technical Reference Manual*.

---

### 11.2.2 SCU speculative coherent requests

This optimization is available for Cortex-R7 MPCore processors only, and only if the L2C-310 Cache Controller is present in the design.

When this feature is enabled, coherent linefill requests are sent speculatively to the L2C-310 Cache Controller in parallel with the SCU tag look-up. If the tag look-up misses, the confirmed linefill is sent to the L2C-310 and gets RDATA earlier because the data request was already initiated by the speculative request. When filtering is enabled, only port 0 can receive speculative linefills.

To support this optimization in the Cortex-R7 MPCore processor:

1. Program the L2C-310 Cache Controller. See the *CoreLink Level 2 Cache Controller (L2C-310) Technical Reference Manual*.
2. Set bit[3] of the SCU Control Register. See [SCU Control Register on page 9-6](#).

---

#### Note

You cannot use this feature when bus ECC is implemented and the L2C-310 Cache Controller is connected.

---

## 11.3 Accessing RAMs using the AXI3 interface

This section describes how to access the TCM RAMs using the AXI slave interface.

The Cortex-R7 MPCore processor has a single AXI slave port. The port is 64 bits wide and conforms to the AXI standard. Within the AXI standard, the slave port uses the **AWUSERS** and **ARUSERS** each as two separate chip select input signals to enable access to the TCMs as shown in [Table 11-11](#).

**Table 11-11 TCM accesses**

<b>AxUSERS[1:0] value</b>	<b>TCM</b>
00	Instruction TCM of processor 0
01	Data TCM of processor 0
10	Instruction TCM of processor 1
11	Data TCM of processor 1.

The external AXI system must generate the chip select signals. The slave interface routes the access to the required RAM.

[Table 11-12](#) shows the MSB bit for the different TCM RAM sizes.

**Table 11-12 MSB bit for the different TCM RAM sizes**

<b>TCM size</b>	<b>ARADDRS[MSB]</b>
4KB	[11]
8KB	[12]
16KB	[13]
32KB	[14]
64KB	[15]
128KB	[16]

**ARADDRS[16:3]** indicates the address of the doubleword in the TCM that you want to access. If you are accessing a TCM that is smaller than the maximum 128KB, then it is possible to address a doubleword that is outside of the physical size of the TCM.

An access to the TCM RAMs is given a SLVERR error response if:

- It is outside the physical size of the targeted TCM RAM, that is, bits of **ARADDRS[16:MSB+1]** are non-zero.
- There is no TCM present. The mapping of bus addresses to **ARUSERS** and **ARADDRS** is determined when the processor is integrated. You must understand this mapping to use the AXI-slave interface in your system.

## 11.4 STRT instructions

Take particular care with non-cacheable write accesses when using the STRT instruction. To put the correct information on the external bus ensure one of the following:

- The access is to Strongly-ordered memory.  
This ensures that the STRT instruction does not merge in the store buffer.
- The access is to Device memory.  
This ensures that the STRT instruction does not merge in the store buffer.
- A DSB instruction is issued before the STRT and after the STRT.  
This prevents an STRT from merging into an existing slot at the same 64-bit address, or merging with another write at the same 64-bit address.

Table 11-13 shows Cortex-R7 processor modes and corresponding **AxPROT** values.

**Table 11-13 Cortex-R7 processor mode and AxPROT values**

Processor mode	Type of access	Value of AxPROT
User	Cacheable read access	User
Privileged		Privileged
User	Non-cacheable read access	User
Privileged		Privileged
-	Cacheable write access	Always marked as Privileged
User	Non-cacheable write access	User
Privileged	Non-cacheable write access	Privileged, except when using STRT



## 11.5 Event communication with an external agent using WFE/SEV

A peripheral connected on the coherency port or any other external agent can participate in the WFE/SEV event communication of the Cortex-R7 MPCore processor by using the **EVENTI** input. When this input is asserted, it sends an event message to all the processors in the design. This is similar to executing a SEV instruction on one processor of the Cortex-R7 MPCore processor. This enables the external agent to signal to the processors that it has released a semaphore and that the processors can leave the power saving mode. The **EVENTI** input must remain HIGH at least one **CLK** clock cycle to be visible by the processors.

The external agent can see that at least one of the processors has executed an SEV instruction by checking the **EVENTO** output. This output is set HIGH for one **CLK** clock cycle when any of the processors executes an SEV instruction.

For more information, see [Power management on page 2-13](#).

## 11.6 Accelerator Coherency Port interface

The optional *Accelerator Coherency Port* (ACP) provides memory coherency between each processor in the Cortex-R7 MPCore processor design and an external master.

The ACP is 64 bits wide, and conforms to the AMBA 3 AXI standard as described in the *AMBA AXI Protocol Specification*.

The following sections describe the ACP:

- [ACP requests](#).
- [ACP limitations on page 11-13](#).

### 11.6.1 ACP requests

The read and write requests performed on the ACP behave differently depending on whether the request is coherent or not. ACP requests behavior is as follows:

#### ACP coherent read requests

An ACP read request is coherent when **ARUSER[0]** = 1 and **ARCACHE[1]** = 1, and **ARVALID** is asserted.

In this case, the SCU enforces coherency.

When the data is present in one of the processors in the Cortex-R7 MPCore design, the data is read directly from the relevant processor, and returned to the ACP port.

When the data is not present in any of the processors, the read request is issued on one of the AXI3 master ports, along with all its AXI parameters, with the exception of the locked attribute.

#### ACP non-coherent read requests

An ACP read request is non-coherent when **ARUSER[0]** = 0 or **ARCACHE[1]** = 0, and **ARVALID** is asserted.

In this case, the SCU does not enforce coherency, and the read request is directly forwarded to one of the available AXI3 master ports.

#### ACP coherent write requests

An ACP write request is coherent when **AWUSER[0]** = 1 and **AWCACHE[1]** = 1, and **AWVALID** is asserted.

In this case, the SCU enforces coherency.

When the data is present in one of the processors in the Cortex-R7 MPCore design, the data is first cleaned and invalidated from the relevant processor.

When the data is not present in any of the processors, or when it has been cleaned and invalidated, the write request is issued on one of the AXI3 master ports, along with all corresponding AXI3 parameters except for the locked attribute.

#### ACP non-coherent write requests

An ACP write request is non-coherent when **AWUSER[0]** = 0 or **AWCACHE[1]** = 0, and **AWVALID** is asserted.

In this case, the SCU does not enforce coherency, and the write request is forwarded directly to one of the available AXI3 master ports.

## 11.6.2 ACP limitations

The ACP is optimized for cache-line length transfers and supports a wide range of AMBA 3 AXI3 requests, but it has some limitations that must be considered:

- [ACP performance limitations](#).
- [ACP functional limitations](#).

### ACP performance limitations

ACP accesses are optimized for transfers that match Cortex-R7 MPCore processor coherent requests:

- A wrapped burst of four doublewords (length = 3, size = 3), with a 64-bit aligned address, and all byte strobes set.
- An incremental burst of four doublewords, with the first address corresponding to the start of a cache line, and all byte strobes set.

For maximum performance use ACP accesses that match this optimized format. ACP accesses that do not match this format cannot benefit from the SCU optimizations, and have significantly lower performance. See [ACP bridge on page 9-40](#) for more information.

### ACP functional limitations

The ACP is a full AMBA3 slave component, with the exception of the following transfers that are not supported:

- Exclusive read and write transactions to coherent memory.
- Locked read and write transactions to coherent memory.
- Optimized coherent read and write transfers when byte strobes are not all set.

Because of this, it is not possible to use the LDREX/STREX mechanism through the ACP to gain exclusive access to coherent memory regions that are marked with **AxUSER[0] = 1** and **AxCACHE[1] = 1**.

However, the LDREX/STREX mechanism is fully supported through the ACP for non-coherent memory regions, marked with **AxUSER[0] = 0** or **AxCACHE[1] = 0**.

See [ACP bridge on page 9-40](#) for more information on access support.

64-bit accesses to the AXI peripheral port always abort. 32-bit wide normal memory non-cacheable accesses from the ACP to the AXI peripheral port do not abort. See [AXI peripheral port on page 2-20](#) for more information.

# Appendix A

## Signal Descriptions

This appendix describes the Cortex-R7 MPCore processor signals. It contains the following sections:

- *About the signal descriptions on page A-2.*
- *Clock and control signals on page A-3.*
- *Reset signals on page A-5.*
- *Interrupt controller signals on page A-6.*
- *Configuration signals on page A-7.*
- *Standby signals on page A-9.*
- *Power management signals on page A-10.*
- *AXI3 interfaces on page A-11.*
- *Performance monitoring signals on page A-23.*
- *Exception flag signals on page A-24.*
- *Error detection notification signals on page A-25.*
- *Test interface on page A-31.*
- *MBIST interface on page A-32.*
- *External debug signals on page A-34.*
- *ETM/ATB interface signals on page A-37.*
- *Memory reconstruction port signals on page A-39.*
- *Power gating interface signals on page A-40.*

## A.1 About the signal descriptions

The tables in this appendix list the Cortex-R7 MPCore processor signals, with their direction, either input or output, and a high-level description.

Many signal names end with [N:0]. The value of N is one less than the number of processors in your design.

There is no link between the number of processors in a design and the number of AXI masters in a design. A single processor design can have one or two AXI master ports.

## A.2 Clock and control signals

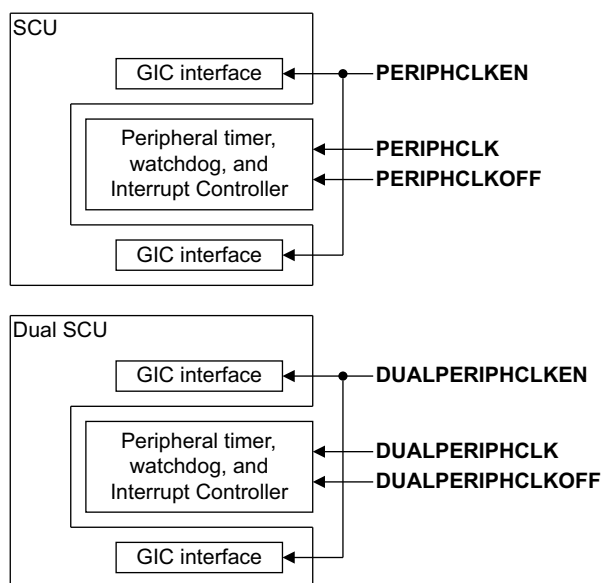
Table A-1 shows the clock and clock control signals.

**Table A-1 Clock and clock control signals**

Name	Type	Source/destination	Description
<b>CLK</b>	Input	Clock controller	Global clock.
<b>CPUCLKOFF[N:0]<sup>a</sup></b>	Input	Reset controller	Individual processor clock control, active-LOW: 0 Clock is enabled. 1 Clock is stopped. This includes the FPU if it is present.
<b>DBGCLKOFF[N:0]<sup>a</sup></b>	Input		Individual processor debug clock control, active-LOW: 0 Processor debug clock is enabled. 1 Processor debug clock is stopped.
<b>DUALPERIPHCLK<sup>b</sup></b>	Input	Clock controller	Clock for dual SCU.
<b>DUALPERIPHCLKEN<sup>b</sup></b>	Input	Reset controller	Clock enable for dual SCU and peripheral interface signals.
<b>DUALPERIPHCLKOFF<sup>ab</sup></b>	Input		Individual processor clock control for timer, watchdog, and interrupt controller of dual SCU.
<b>PERIPHCLK</b>	Input	Clock controller	Clock for the timer, watchdog, and interrupt controller.
<b>PERIPHCLKEN</b>	Input	Reset controller	Clock enable for SCU and peripheral interface signals.
<b>PERIPHCLKOFF<sup>ab</sup></b>	Input		Individual processor clock control for timer, watchdog, and interrupt controller of SCU.
<b>SCUCLKOFF<sup>ab</sup></b>	Input		Clock control delay for dual SCU.
<b>CTCLKOFF<sup>a</sup></b>	Input		Used to control the CoreSight debug logic clock, that is, CTI0, CTI1, CTM, and ROM table.
<b>ETM0CLKOFF<sup>a</sup></b>	Input		Used to control the ETM0 clock, if ETM0 is present.
<b>ETM1CLKOFF<sup>a</sup></b>	Input		Used to control the ETM1 clock, if ETM1 is present.

a. Used to deassert the reset synchronously when leaving reset, but not used for clock enable.

b. Only present if lock-step or split/lock is implemented. Figure A-1 on page A-4 shows how these signals are used in a single SCU and a dual SCU implementation.



**Figure A-1 Clocking in lock-step or split/lock implementation**

See [Clocking, resets, and initialization on page 2-5](#).

## A.3 Reset signals

Table A-2 shows the reset signals.

**Table A-2 Reset signals**

Name	Type	Source/destination	Description
<b>nCPURESET[N:0]</b>	Input	Reset controller	Individual processor resets.
<b>nCPUHALT[N:0]</b>	Input		Individual processor input. It can be asserted while the processor is in reset to stop the processor from fetching and executing instructions after coming out of reset. While the processor is halted in this way, the TCMs can be preloaded with the appropriate data. When it is deasserted, the processor starts fetching instructions from the reset vector address in the normal way.
<b>nDBGRESET[N:0]</b>	Input		Processor debug logic resets.
<b>nPERIPHRESET</b>	Input		Timer and interrupt controller reset.
<b>nSCURESET</b>	Input		SCU global reset.
<b>nCTRESET</b>	Input		Reset for CoreSight debug logic, that is, CTI0, CTI1, CTM, and ROM table.
<b>nETM0RESET</b>	Input		Reset for ETM0, if present.
<b>nETM1RESET</b>	Input		Reset for ETM1, if present.
<b>nWDRESET[N:0]</b>	Input		Processor watchdog resets.
<b>WDRESETREQ[N:0]</b>	Output		Processor watchdog reset requests.

See [Resets](#) on page 2-5.



## A.4 Interrupt controller signals

Table A-3 shows the interrupt controller signals.

**Table A-3 Interrupt controller signals**

Name	Type	Source/destination	Description
<b>IRQS[x:0]</b>	Input	Interrupt sources	Interrupt distributor interrupt lines. x can be 31, 63, ..., up to 479 in increments of 32. If there are no interrupt lines, this input is removed.
<b>nFIQ[N:0]</b>	Input		Individual processor legacy FIQ request input lines. Active-LOW interrupt request: 0                   Active interrupt. 1                   Do not activate interrupt. The processor treats the <b>nFIQ</b> input as level sensitive. The <b>nFIQ</b> input must be asserted until the processor acknowledges the interrupt.
<b>nFIQOUT[N:0]</b>	Output	Power controller	Active-LOW <b>FIQ</b> outputs from the internal GIC to processor 1 and processor 0 as appropriate. These indicate when interrupts are being forwarded to the processor.
<b>nIRQ[N:0]</b>	Input	Interrupt sources	Individual processor legacy IRQ request input lines. Active-LOW interrupt request: 0                   Active interrupt. 1                   Do not activate interrupt. The processor treats the <b>nIRQ</b> input as level sensitive. The <b>nIRQ</b> input must be asserted until the processor acknowledges the interrupt.
<b>nIRQOUT[N:0]</b>	Output	Power controller	Active-LOW <b>IRQ</b> outputs from the internal GIC to processor 1 and processor 0 as appropriate. These indicate when interrupts are being forwarded to the processor.

## A.5 Configuration signals

Table A-4 shows the configuration signals.

**Table A-4 Configuration signals**

Name	Type	Source/destination	Description
<b>AXIPARITYLEVEL<sup>a</sup></b>	Input	System configuration	<p>Selects between odd and even parity for buses:</p> <p>0 Even parity.</p> <p>1 Odd parity.</p>
<b>CFGEND[N:0]</b>	Input		<p>Individual processor endianness configuration.</p> <p>Forces the EE bit in the CP15 c1 Control Register (SCTLR) to 1 at reset so that the processor boots with big-endian data handling:</p> <p>0 EE bit is LOW.</p> <p>1 EE bit is HIGH.</p> <p>This input is only sampled during reset of the processor.</p> <p>See <a href="#">System Control Register</a> on page 4-23.</p>
<b>CFGNMFI[N:0]</b>	Input		<p>Configuration of FIs to be nonmaskable:</p> <p>0 Clear the NMFI bit in the CP15 c1 Control Register.</p> <p>1 Set the NMFI bit in the CP15 c1 Control Register.</p> <p>This input is only sampled during reset of the processor.</p> <p>See <a href="#">System Control Register</a> on page 4-23.</p>
<b>CLUSTERID[3:0]</b>	Input		<p>Value read in Cluster ID field, bits[11:8], of the <i>Multiprocessor Affinity Register</i> (MPIDR).</p> <p>See <a href="#">Multiprocessor Affinity Register</a> on page 4-18.</p>
<b>INITRAM0</b>	Input		<p>Input present if TCM present for processor 0. It enables the processor 0 to boot from the Instruction TCM. This input, when tied HIGH, enables the instruction TCM on leaving reset.</p> <p>See <a href="#">ITCM Region Register</a> on page 4-34.</p>
<b>INITRAM1</b>	Input		<p>Input present if TCM present for processor 1. It enables the processor 1 to boot from the Instruction TCM. This input, when tied HIGH, enables the instruction TCM on leaving reset.</p> <p>See <a href="#">ITCM Region Register</a> on page 4-34.</p>
<b>MFILTEREN</b>	Input		<p>For use with configurations with two master ports.</p> <p>It enables filtering of address ranges at reset for AXI Master port 1. This signal is sampled on exit from reset and sets the default value of the MFILTEREN bit in the SCU Control Register. See <a href="#">SCU Control Register</a> on page 9-6.</p> <p>0 Address filtering off.</p> <p>1 Address filtering on.</p> <p>See <a href="#">SCU Control Register</a> on page 9-6 and <a href="#">AXI master port 1</a> on page 2-19.</p>
<b>MFILTEREND[11:0]</b>	Input		<p>For use with configurations with two master ports.</p> <p>Specifies the end address for address filtering at reset on AXI master port 1.</p> <p>See <a href="#">SCU Control Register</a> on page 9-6 and <a href="#">AXI master port 1</a> on page 2-19.</p>

Table A-4 Configuration signals (continued)

Name	Type	Source/destination	Description
<b>MFILTERSTART[11:0]</b>	Input	System configuration	<p>For use with configurations with two master ports. Specifies the start address for address filtering at reset on AXI master port 1.</p> <p>See <a href="#">SCU Control Register on page 9-6</a> and <a href="#">AXI master port 1 on page 2-19</a>.</p>
<b>PERIPHBASE[31:13]</b>	Input		<p>Specifies the base address for timers, watchdogs, interrupt controller, and SCU registers. Only accessible with memory-mapped accesses.</p> <p>This value can be retrieved by a processor using the Configuration Base Address Register. See <a href="#">Configuration Base Address Register on page 4-42</a>.</p> <p>———— <b>Note</b> ————</p> <p>This address must be in the range defined by <b>PFILTERSTART[31:20]</b> and <b>PFILTEREND[31:20]</b>.</p>
<b>PFILTEREND[11:0]</b>	Input		<p>For use with configurations with the AXI Master peripheral port. Specifies the end address for address filtering at reset on the AXI peripheral port.</p> <p>See <a href="#">AXI peripheral port on page 2-20</a>.</p>
<b>PFILTERSTART[11:0]</b>	Input		<p>For use with configurations with the AXI Master peripheral port. Specifies the start address for address filtering at reset on the AXI peripheral port.</p> <p>See <a href="#">AXI peripheral port on page 2-20</a>.</p>
<b>SMPnAMP[N:0]</b>	Output	System integrity controller	<p>Indicates AMP or SMP mode for each processor:</p> <p>0            Asymmetric.</p> <p>1            Symmetric.</p> <p>This output reflects the value of ACTLR.SMP.</p> <p>See <a href="#">Auxiliary Control Register on page 4-25</a>.</p>
<b>TEINIT[N:0]</b>	Input	System configuration	<p>Individual processor out-of-reset default exception handling state:</p> <p>0            ARM.</p> <p>1            Thumb.</p> <p>This input is only sampled during reset of the processor. It sets the initial value of SCTLR.TE.</p> <p>See <a href="#">System Control Register on page 4-23</a>.</p>
<b>VINITHI[N:0]</b>	Input		<p>Individual processor control of the location of the exception vectors at reset:</p> <p>0            Exception vectors start at address 0x00000000.</p> <p>1            Exception vectors start at address 0xFFFF0000.<sup>b</sup></p> <p>This input is only sampled during reset of the processor. It sets the initial value of SCTLR.V.</p> <p>See <a href="#">System Control Register on page 4-23</a> and <a href="#">ITCM Region Register on page 4-34</a>.</p>

a. Only present if bus ECC is selected. This is a build option.

b. `HIVECS == 1` is deprecated in PMSAv7.

## A.6 Standby signals

[Table A-5](#) shows the Standby and Wait for Event signals.

**Table A-5 Standby and Wait for event signals**

Name	Type	Source/destination	Description
STANDBYWFI[N:0]	Output	Wakeup controller	Indicates if processor 1 or processor 0 is in Standby WFI.
STANDBYWFE[N:0]	Output		Indicates if processor 1 or processor 0 is in Standby WFE.

[Table A-6](#) shows the event signals.

**Table A-6 Event signals**

Name	Type	Source/destination	Description
EVENTI	Input	External coherent agent	Macrocell standby and wait for event signal, event input.
EVENTO	Output		Macrocell standby and wait for event signal, event output.

See [Standby modes](#) on page 2-14.

For SCU Standby, see [SCU Control Register](#) on page 9-6.

## A.7 Power management signals

Table A-7 shows the power management signals.

**Table A-7 Power management signals**

Name	Type	Source/destination	Description
<b>PWRCTLO0[1:0]</b>	Output	Power controller	Reset value for processor 0 status field, bits[1:0] of SCU CPU Power Status Register.
<b>PWRCTLO1[1:0]<sup>a</sup></b>	Output		Reset value for processor 1 status field, bits[9:8] of SCU CPU Power Status Register.
<b>SCUIDLE</b>	Output	L2C-310 or power controller	<p>In the case of the L2C-310, the <b>SCUIDLE</b> output can be connected to the <b>STOPCLK</b> input of the L2C-310. Indicates the SCU is idle. The SCU is idle when both processors are in WFI or in powerdown, and there is no pending transaction in the SCU on any of the AXI ports, that is, ACP, AXI master 0, AXI master 1, or AXI peripheral port.</p> <p>———— <b>Note</b> ————</p> <p>When using the ACP, even if there is no activity on the bus, the <b>ACLKENS</b> input must remain HIGH, or must toggle at least once through HIGH, after the activity has stopped. If not, the <b>SCUIDLE</b> output cannot go HIGH.</p>

a. Only present if processor 1 is present.

See [Power domains](#) on page 2-18.

For SCU power management, see [SCU CPU Power Status Register](#) on page 9-9.

## A.8 AXI3 interfaces

This section describes the following AXI3 interfaces:

- [AXI master signals](#).
- [AXI peripheral port signals on page A-15](#).
- [AXI ACP slave port signals on page A-18](#).
- [AXI TCM slave port signals on page A-20](#).

### A.8.1 AXI master signals

**x** in the signal name represents either master 0, the primary AXI master, or master 1, the optional secondary AXI master.

[Table A-8](#) shows the AXI master interface clock enable signals.

**Table A-8 AXI master interface clock enable signals**

Name	Type	Source/destination	Description
<b>INCLKENM<sub>x</sub></b>	Input	CLK	<p>Clock bus enable for the AXI bus that enables the AXI interface to operate at either:</p> <ul style="list-style-type: none"> <li>• Integer ratios of the system clock.</li> <li>• Half integer ratios of the system clock.</li> </ul> <p>Inputs are sampled on rising edges of <b>CLK</b> only when <b>INCLKENM<sub>x</sub></b> is HIGH.</p>
<b>OUTCLKENM<sub>x</sub></b>	Input		<p>Clock bus enable for the AXI bus that enables the AXI interface to operate at either:</p> <ul style="list-style-type: none"> <li>• Integer ratios of the system clock.</li> <li>• Half integer ratios of the system clock.</li> </ul> <p>Outputs are updated on rising edges of <b>CLK</b> only when <b>OUTCLKENM<sub>x</sub></b> is HIGH.</p>

[Table A-9](#) shows the AXI master read address signals.

**Table A-9 AXI master read address signals**

Name	Type	Source/destination	Description
<b>ARADDRM<sub>x</sub>[31:0]</b>	Output	AXI3 device	Address.
<b>ARBURSTM<sub>x</sub>[1:0]</b>	Output		<p>Burst type:</p> <p>0b1 INCR incrementing burst.</p> <p>0b10 WRAP wrapping burst.</p> <p>All other values are reserved.</p>
<b>ARCAHEM<sub>x</sub>[3:0]</b>	Output		Cache type giving additional information about cacheable characteristics.
<b>ARIDM<sub>x</sub>[n]<sup>a</sup></b>	Output		Request ID.

Table A-9 AXI master read address signals (continued)

Name	Type	Source/destination	Description																																
ARLENMx[3:0]	Output	AXI3 device	<div>The number of data transfers that can occur within each burst. Cacheable traffic generates transactions with four data transfers. For a description of other traffic, see <a href="#">Supported AXI3 transfers on page 11-3</a>. Burst transactions from the ACP can be 1-16 transfers long.</div> <table><tr><td>0b0000</td><td>1 data transfer.</td></tr><tr><td>0b0001</td><td>2 data transfers.</td></tr><tr><td>0b0010</td><td>3 data transfers.</td></tr><tr><td>0b0011</td><td>4 data transfers.</td></tr><tr><td>0b0100</td><td>5 data transfers.</td></tr><tr><td>0b0101</td><td>6 data transfers.</td></tr><tr><td>0b0110</td><td>7 data transfers.</td></tr><tr><td>0b0111</td><td>8 data transfers.</td></tr><tr><td>0b1000</td><td>9 data transfers.</td></tr><tr><td>0b1001</td><td>10 data transfers.</td></tr><tr><td>0b1010</td><td>11 data transfers.</td></tr><tr><td>0b1011</td><td>12 data transfers.</td></tr><tr><td>0b1100</td><td>13 data transfers.</td></tr><tr><td>0b1101</td><td>14 data transfers.</td></tr><tr><td>0b1110</td><td>15 data transfers.</td></tr><tr><td>0b1111</td><td>16 data transfers.</td></tr></table>	0b0000	1 data transfer.	0b0001	2 data transfers.	0b0010	3 data transfers.	0b0011	4 data transfers.	0b0100	5 data transfers.	0b0101	6 data transfers.	0b0110	7 data transfers.	0b0111	8 data transfers.	0b1000	9 data transfers.	0b1001	10 data transfers.	0b1010	11 data transfers.	0b1011	12 data transfers.	0b1100	13 data transfers.	0b1101	14 data transfers.	0b1110	15 data transfers.	0b1111	16 data transfers.
0b0000	1 data transfer.																																		
0b0001	2 data transfers.																																		
0b0010	3 data transfers.																																		
0b0011	4 data transfers.																																		
0b0100	5 data transfers.																																		
0b0101	6 data transfers.																																		
0b0110	7 data transfers.																																		
0b0111	8 data transfers.																																		
0b1000	9 data transfers.																																		
0b1001	10 data transfers.																																		
0b1010	11 data transfers.																																		
0b1011	12 data transfers.																																		
0b1100	13 data transfers.																																		
0b1101	14 data transfers.																																		
0b1110	15 data transfers.																																		
0b1111	16 data transfers.																																		
ARLOCKMx[1:0]	Output		<div>Lock type:</div> <table><tr><td>0b00</td><td>Normal access.</td></tr><tr><td>0b01</td><td>Exclusive access.</td></tr><tr><td>0b10</td><td>Locked access.</td></tr></table>	0b00	Normal access.	0b01	Exclusive access.	0b10	Locked access.																										
0b00	Normal access.																																		
0b01	Exclusive access.																																		
0b10	Locked access.																																		
ARPROTMx[2:0]	Output		Protection type																																
ARREADYMx	Input		Address ready																																
ARSIZEMx[1:0]	Output		<div>Burst size:</div> <table><tr><td>0b000</td><td>8-bit transfer.</td></tr><tr><td>0b001</td><td>16-bit transfer.</td></tr><tr><td>0b010</td><td>32-bit transfer.</td></tr><tr><td>0b011</td><td>64-bit transfer.</td></tr></table>	0b000	8-bit transfer.	0b001	16-bit transfer.	0b010	32-bit transfer.	0b011	64-bit transfer.																								
0b000	8-bit transfer.																																		
0b001	16-bit transfer.																																		
0b010	32-bit transfer.																																		
0b011	64-bit transfer.																																		
ARUSERMx[8:0]	Output		Transfer attributes. See <a href="#">AXI3 USER bits on page 11-3</a> .																																
ARVALIDMx	Output		Address valid.																																

- a. You can define the number of AXI ID bits on this port using the AXISC\_ID\_BIT build parameter. If the ACP is implemented, [n] is [AXISC\_ID\_BIT:0], that is, the number of ACP ID bits + 1. If the ACP is not implemented, [n] is [4:0].

Table A-10 shows the AXI master read data signals.

**Table A-10 AXI master read data signals**

Name	Type	Source/destination	Description
<b>RDATA<sub>m</sub>[63:0]</b>	Input	AXI3 device	Read data
<b>RDATAERRCODE<sub>m</sub>[7:0]</b>	Input		ECC bits on data bus, when BUS_ECC build parameter is set
<b>RID<sub>m</sub>[n]<sup>a</sup></b>	Input		Read ID
<b>RLAST<sub>m</sub></b>	Input		Read last indication
<b>RREADY<sub>m</sub></b>	Output		Read ready
<b>RRESP<sub>m</sub>[1:0]</b>	Input		Read response
<b>RVALID<sub>m</sub></b>	Input		Read valid
<b>SREND<sub>m</sub>[3:0]</b>	Input		Speculative read information from optional L2 Cache Controller. See the <i>CoreLink Level 2 Cache Controller (L2C-310) Technical Reference Manual</i> for more information.
<b>SRID<sub>m</sub>[n]<sup>a</sup></b>	Input		ID for speculative reads returned by L2 Cache Controller

- a. You can define the number of AXI ID bits on this port using the AXISC\_ID\_BIT build parameter. If the ACP is implemented, [n] is [AXISC\_ID\_BIT:0], that is, the number of ACP ID bits + 1. If the ACP is not implemented, [n] is [4:0].

Table A-11 shows the AXI master write address signals.

**Table A-11 AXI master write address signals**

Name	Type	Source/destination	Description
<b>AWADDR<sub>m</sub>[31:0]</b>	Output	AXI3 device	Address
<b>AWBURST<sub>m</sub>[1:0]</b>	Output		Burst type: 0b1 INCR incrementing burst. 0b10 WRAP wrapping burst. All other values are reserved.
<b>AWCACHE<sub>m</sub>[3:0]</b>	Output		Cache type giving additional information about cacheable characteristics.
<b>AWID<sub>m</sub>[n]<sup>a</sup></b>	Output		Write ID



Table A-11 AXI master write address signals (continued)

Name	Type	Source/destination	Description																																
AWLENMx[3:0]	Output	AXI3 device	<p>The number of data transfers that can occur within each burst. For a description of other processor-generated traffic, see <a href="#">Supported AXI3 transfers on page 11-3</a>. Burst transactions from the ACP can be 1-16 transfers long.</p> <table><tr><td>0b000</td><td>1 data transfer.</td></tr><tr><td>0b001</td><td>2 data transfers.</td></tr><tr><td>0b010</td><td>3 data transfers.</td></tr><tr><td>0b011</td><td>4 data transfers.</td></tr><tr><td>0b100</td><td>5 data transfers.</td></tr><tr><td>0b101</td><td>6 data transfers.</td></tr><tr><td>0b110</td><td>7 data transfers.</td></tr><tr><td>0b111</td><td>8 data transfers.</td></tr><tr><td>0b1000</td><td>9 data transfers.</td></tr><tr><td>0b1001</td><td>10 data transfers.</td></tr><tr><td>0b1010</td><td>11 data transfers.</td></tr><tr><td>0b1011</td><td>12 data transfers.</td></tr><tr><td>0b1100</td><td>13 data transfers.</td></tr><tr><td>0b1101</td><td>14 data transfers.</td></tr><tr><td>0b1110</td><td>15 data transfers.</td></tr><tr><td>0b1111</td><td>16 data transfers.</td></tr></table>	0b000	1 data transfer.	0b001	2 data transfers.	0b010	3 data transfers.	0b011	4 data transfers.	0b100	5 data transfers.	0b101	6 data transfers.	0b110	7 data transfers.	0b111	8 data transfers.	0b1000	9 data transfers.	0b1001	10 data transfers.	0b1010	11 data transfers.	0b1011	12 data transfers.	0b1100	13 data transfers.	0b1101	14 data transfers.	0b1110	15 data transfers.	0b1111	16 data transfers.
0b000	1 data transfer.																																		
0b001	2 data transfers.																																		
0b010	3 data transfers.																																		
0b011	4 data transfers.																																		
0b100	5 data transfers.																																		
0b101	6 data transfers.																																		
0b110	7 data transfers.																																		
0b111	8 data transfers.																																		
0b1000	9 data transfers.																																		
0b1001	10 data transfers.																																		
0b1010	11 data transfers.																																		
0b1011	12 data transfers.																																		
0b1100	13 data transfers.																																		
0b1101	14 data transfers.																																		
0b1110	15 data transfers.																																		
0b1111	16 data transfers.																																		
AWLOCKMx[1:0]	Output		<p>Lock type:</p> <table><tr><td>0b0</td><td>Normal access.</td></tr><tr><td>0b1</td><td>Exclusive access.</td></tr><tr><td>0b10</td><td>Locked access.</td></tr></table>	0b0	Normal access.	0b1	Exclusive access.	0b10	Locked access.																										
0b0	Normal access.																																		
0b1	Exclusive access.																																		
0b10	Locked access.																																		
AWPROTMx[2:0]	Output		Protection type																																
AWREADYMx	Input		Address ready																																
AWSIZEMx[1:0]	Output		<p>Burst size:</p> <table><tr><td>0b00</td><td>8-bit transfer.</td></tr><tr><td>0b01</td><td>16-bit transfer.</td></tr><tr><td>0b10</td><td>32-bit transfer.</td></tr><tr><td>0b11</td><td>64-bit transfer.</td></tr></table>	0b00	8-bit transfer.	0b01	16-bit transfer.	0b10	32-bit transfer.	0b11	64-bit transfer.																								
0b00	8-bit transfer.																																		
0b01	16-bit transfer.																																		
0b10	32-bit transfer.																																		
0b11	64-bit transfer.																																		
AWUSERMx[10:0]	Output		Transfer attributes. See <a href="#">AXI3 USER bits on page 11-3</a> .																																
AWVALIDMx	Output		Address valid																																

- a. You can define the number of AXI ID bits on this port using the AXISC\_ID\_BIT build parameter. If the ACP is implemented, [n] is [AXISC\_ID\_BIT:0], that is, the number of ACP ID bits + 1. If the ACP is not implemented, [n] is [4:0].

Table A-12 shows the AXI master write data signals.

**Table A-12 AXI master write data signals**

Name	Type	Source/destination	Description
<b>WDATAMx[63:0]</b>	Output	AXI3 device	Write data
<b>WIDMx[n]<sup>a</sup></b>	Output		Write ID
<b>WLASTMx</b>	Output		Write last indication
<b>WREADYMx</b>	Input		Write ready
<b>WSTRBMx[7:0]</b>	Output		Write byte-lane strobe
<b>WVALIDMx</b>	Output		Write valid
<b>WUSERMx[1:0]</b>	Output		Transfer attributes. See <i>AXI3 USER bits</i> on page 11-3.

- a. You can define the number of AXI ID bits on this port using the `AXISC_ID_BIT` build parameter. If the ACP is implemented, `[n]` is `[AXISC_ID_BIT:0]`, that is, the number of ACP ID bits + 1. If the ACP is not implemented, `[n]` is `[4:0]`.

Table A-13 shows the AXI master write response signals.

**Table A-13 AXI master write response signals**

Name	Type	Source/destination	Description
<b>BIDMx[n]<sup>a</sup></b>	Input	L2C-310 or other system AXI3 devices	Response ID
<b>BREADYMx</b>	Output		Response ready
<b>BRESPMx[1:0]</b>	Input		Write response
<b>BVALIDMx</b>	Input		Response valid

- a. You can define the number of AXI ID bits on this port using the `AXISC_ID_BIT` build parameter. If the ACP is implemented, `[n]` is `[AXISC_ID_BIT:0]`, that is, the number of ACP ID bits + 1. If the ACP is not implemented, `[n]` is `[4:0]`.

## A.8.2 AXI peripheral port signals

Table A-14 shows the AXI peripheral port clock enable signals.

**Table A-14 Clock enable signals**

Name	Type	Source/destination	Description
<b>INCLKENMP</b>	Input	CLK	Clock enable
<b>OUTCLKENMP</b>	Input		Clock enable

Table A-15 shows the AXI peripheral port read address signals.

**Table A-15 AXI peripheral port read address signals**

Name	Type	Source/destination	Description
<b>ARADDRMP[31:0]</b>	Output	AXI3 device	Address.
<b>ARBURSTMP[1:0]</b>	Output		Burst type.
<b>ARCACHEMP[3:0]</b>	Output		Cache type.
<b>ARIDMP[n]<sup>a</sup></b>	Output		Address ID.
<b>ARLENMP[3:0]</b>	Output		Burst length.
<b>ARLOCKMP[1:0]</b>	Output		Lock type.
<b>ARPROTMP[2:0]</b>	Output		Protection type.
<b>ARREADYMP</b>	Input		Address ready.
<b>ARSIZEMP[1:0]</b>	Output		Burst size.
<b>ARUSERMP[8:0]</b>	Output		Transfer attributes. See <i>AXI3 USER bits</i> on page 11-3.
<b>ARVALIDMP</b>	Output		Address valid.

- a. You can define the number of AXI ID bits on this port using the `AXISC_ID_BIT` build parameter. If the ACP is implemented, [n] is [AXISC\_ID\_BIT:0], that is, the number of ACP ID bits + 1. If the ACP is not implemented, [n] is [4:0].

Table A-16 shows the AXI peripheral port read data signals.

**Table A-16 AXI peripheral port read data signals**

Name	Type	Source/destination	Description
<b>RVALIDMP</b>	Input	AXI3 device	Read valid
<b>RREADYMP</b>	Output		Read ready
<b>RIDMP[n]<sup>a</sup></b>	Input		Read ID
<b>RLASTMP</b>	Input		Read last
<b>RDATAMP[31:0]</b>	Input		Read data
<b>RRESPMP[1:0]</b>	Input		Read response

- a. You can define the number of AXI ID bits on this port using the `AXISC_ID_BIT` build parameter. If the ACP is implemented, [n] is [AXISC\_ID\_BIT:0], that is, the number of ACP ID bits + 1. If the ACP is not implemented, [n] is [4:0].

Table A-17 shows the AXI peripheral port write address signals.

**Table A-17 AXI peripheral port write address signals**

Name	Type	Source/destination	Description
<b>AWVALIDMP</b>	Output	AXI3 device	Address valid.
<b>AWREADYMP</b>	Input		Address ready.
<b>AWIDMP[n]<sup>a</sup></b>	Output		Address ID.
<b>AWADDRMP[31:0]</b>	Output		Address.
<b>AWSIZEMP[1:0]</b>	Output		Burst size.
<b>AWLENMP[3:0]</b>	Output		Burst length.
<b>AWBURSTMP[1:0]</b>	Output		Burst type.
<b>AWCACHEMP[3:0]</b>	Output		Cache type.
<b>AWPROTMP[2:0]</b>	Output		Protection type.
<b>AWLOCKMP[1:0]</b>	Output		Lock type.
<b>AWUSERMP[10:0]</b>	Output		Transfer attributes. See <i>AXI3 USER bits</i> on page 11-3.

- a. You can define the number of AXI ID bits on this port using the `AXISC_ID_BIT` build parameter. If the ACP is implemented, [n] is [AXISC\_ID\_BIT:0], that is, the number of ACP ID bits + 1. If the ACP is not implemented, [n] is [4:0].

Table A-18 shows the AXI peripheral port write data signals.

**Table A-18 AXI peripheral port write data signals**

Name	Type	Source/destination	Description
<b>WVALIDMP</b>	Output	AXI3 device	Write valid.
<b>WREADYMP</b>	Input		Write ready.
<b>WIDMP[n]<sup>a</sup></b>	Output		Write ID.
<b>WLASTMP</b>	Output		Write last.
<b>WSTRBMP[3:0]</b>	Output		Write strobes.
<b>WDATAMP[31:0]</b>	Output		Write data.
<b>WUSERMP[1:0]</b>	Output		Transfer attributes. See <i>AXI3 USER bits</i> on page 11-3.

- a. You can define the number of AXI ID bits on this port using the `AXISC_ID_BIT` build parameter. If the ACP is implemented, [n] is [AXISC\_ID\_BIT:0], that is, the number of ACP ID bits + 1. If the ACP is not implemented, [n] is [4:0].

Table A-19 shows the AXI peripheral port write response signals.

**Table A-19 AXI peripheral port write response signals**

Name	Type	Source/destination	Description
<b>BVALIDMP</b>	Input	AXI3 device	Response valid
<b>BREADYMP</b>	Output		Response ready
<b>BIDMP[n]<sup>a</sup></b>	Input		Response ID
<b>BRESPMP[1:0]</b>	Input		Write response

- a. You can define the number of AXI ID bits on this port using the `AXISC_ID_BIT` build parameter. If the ACP is implemented, [n] is `[AXISC_ID_BIT:0]`, that is, the number of ACP ID bits + 1. If the ACP is not implemented, [n] is `[4:0]`.

### A.8.3 AXI ACP slave port signals

Table A-20 shows the AXI ACP slave port clock enable signal.

**Table A-20 AXI ACP slave port clock enable signal**

Name	Type	Source/destination	Description
<b>ACLKENS</b>	Input	Clock controller	Clock bus enable for the AXI bus that enables the AXI interface to operate at integer ratios of the system clock.

Table A-21 shows the AXI ACP slave port read address signals.

**Table A-21 AXI ACP slave port read address signals**

Name	Type	Source/destination	Description
<b>ARVALIDSC</b>	Input	AXI3 device	Address valid.
<b>ARREADYSC</b>	Output		Address ready.
<b>ARIDSC[n]<sup>a</sup></b>	Input		Address ID.
<b>ARADDRSC[31:0]</b>	Input		Address.
<b>ARSIZE[1:0]</b>	Input		Burst size.
<b>ARLENSC[3:0]</b>	Input		Burst length.
<b>ARBURSTSC[1:0]</b>	Input		Burst type.
<b>ARCACHESC[3:0]</b>	Input		Cache type.
<b>ARPROTSC[2:0]</b>	Input		Protection type.
<b>ARLOCKSC</b>	Input		Lock type.
<b>ARUSERSC[4:0]</b>	Input		Transfer attributes.

- a. You can define the number of AXI ID bits on this port using the `AXISC_ID_BIT` build parameter.

Table A-22 shows the AXI ACP slave port read data signals.

**Table A-22 AXI ACP slave port read data signals**

Name	Type	Source/destination	Description
<b>RVALIDSC</b>	Output	AXI3 device	Read valid.
<b>RREADYSC</b>	Input		Read ready.
<b>RIDSC[n]<sup>a</sup></b>	Output		Read ID.
<b>RLASTSC</b>	Output		Read last.
<b>RDATASC[63:0]</b>	Output		Read data.
<b>RRESPSC[1:0]</b>	Output		Read response.

a. You can define the number of AXI ID bits on this port using the AXISC\_ID\_BIT build parameter.

Table A-23 shows the AXI ACP slave port write address signals.

**Table A-23 AXI ACP slave port write address signals**

Name	Type	Source/destination	Description
<b>AWVALIDSC</b>	Input	AXI3 device	Address valid.
<b>AWREADYSC</b>	Output		Address ready.
<b>AWIDSC[n]<sup>a</sup></b>	Input		Address ID.
<b>AWADDRSC[31:0]</b>	Input		Address.
<b>AWSIZESC[1:0]</b>	Input		Burst size.
<b>AWLENSC[3:0]</b>	Input		Burst length. The maximum burst transfer must correspond to an L1 cache line, that is, 256 bits.
<b>AWBURSTSC[1:0]</b>	Input		Burst type.
<b>AWCACHESC[3:0]</b>	Input		Cache type.
<b>AWPROTSC[2:0]</b>	Input		Protection type.
<b>AWLOCKSC</b>	Input		Lock type.
<b>AWUSERSC[5:0]</b>	Input		Transfer attributes.

a. You can define the number of AXI ID bits on this port using the AXISC\_ID\_BIT build parameter.

Table A-24 shows the AXI ACP slave port write data signals.

**Table A-24 AXI ACP slave port write data signals**

Name	Type	Source/destination	Description
<b>WVALIDSC</b>	Input	AXI3 device	Write valid.
<b>WREADYSC</b>	Output		Write ready.
<b>WIDSC[n]<sup>a</sup></b>	Input		Write ID.

**Table A-24 AXI ACP slave port write data signals (continued)**

Name	Type	Source/destination	Description
WLASTSC	Input	AXI3 device	Write last.
WSTRBSC[7:0]	Input		Write strobes.
WDATASC[63:0]	Input		Write data.

- a. You can define the number of AXI ID bits on this port using the AXISC\_ID\_BIT build parameter.

Table A-25 shows the AXI ACP slave port write response signals.

**Table A-25 AXI ACP slave port write response signals**

Name	Type	Source/destination	Description
BVALIDSC	Output	AXI3 device	Response valid.
BREADYSC	Input		Response ready.
BIDSC[n] <sup>a</sup>	Output		Response ID.
BRESPSC[1:0]	Output		Write response.

- a. You can define the number of AXI ID bits on this port using the AXISC\_ID\_BIT build parameter.

#### A.8.4 AXI TCM slave port signals

Table A-26 shows the AXI TCM slave port clock enable signal.

**Table A-26 AXI TCM slave port clock enable signal**

Name	Type	Source/destination	Description
ACLKENST	Input	Clock controller	Clock bus enable for the AXI bus that enables the AXI interface to operate at integer ratios of the system clock.

Table A-27 shows the AXI TCM slave port read address signals.

**Table A-27 AXI TCM slave port read address signals**

Name	Type	Source/destination	Description
ARVALIDST	Input	AXI3 device	Address valid
ARREADYST	Output		Address ready
ARIDST[n] <sup>a</sup>	Input		Address ID.
ARADDRST[16:0]	Input		Address
ARSIZEST[1:0]	Input		Burst size
ARLENST[3:0]	Input		Burst length
ARBURSTST[1:0]	Input		Burst type
ARUSERST[1:0]	Input		Transfer attributes

- a. You can define the number of AXI ID bits on this port using the AXIST\_ID\_BIT build parameter.

Table A-28 shows the AXI TCM slave port read data signals.

**Table A-28 AXI TCM slave port read data signals**

Name	Type	Source/destination	Description
RVALIDST	Output	AXI3 device	Read valid
RREADYST	Input		Read ready
RIDST[n] <sup>a</sup>	Output		Read ID
RLASTST	Output		Read last
RDATAST[63:0]	Output		Read data
RRESPST[1:0]	Output		Read response

- a. You can define the number of AXI ID bits on this port using the AXIST\_ID\_BIT build parameter.

Table A-29 shows the AXI TCM slave port write address signals.

**Table A-29 AXI TCM slave port write address signals**

Name	Type	Source/destination	Description
AWVALIDST	Input	AXI3 device	Address valid.
AWREADYST	Output		Address ready.
AWIDST[n] <sup>a</sup>	Input		Address ID.
AWADDRST[16:0]	Input		Address.
AWSIZEST[1:0]	Input		Burst size.
AWLENST[3:0]	Input		Burst length. The maximum burst transfer must correspond to an L1 cache line, that is, 256 bits.
AWBURSTST[1:0]	Input		Burst type.
AWUSERST[1:0]	Input		Transfer attributes.

- a. You can define the number of AXI ID bits on this port using the AXIST\_ID\_BIT build parameter.



Table A-30 shows the AXI TCM slave port write data signals.

**Table A-30 AXI TCM slave port write data signals**

Name	Type	Source/destination	Description
<b>WVALIDST</b>	Input	AXI3 device	Write valid
<b>WREADYST</b>	Output		Write ready
<b>WIDST[n]<sup>a</sup></b>	Input		Write ID
<b>WLASTST</b>	Input		Write last
<b>WSTRBST[7:0]</b>	Input		Write strobes
<b>WDATAST[63:0]</b>	Input		Write data

a. You can define the number of AXI ID bits on this port using the AXIST\_ID\_BIT build parameter.

Table A-31 shows the AXI TCM slave port write response signals.

**Table A-31 AXI TCM slave port write response signals**

Name	Type	Source/destination	Description
<b>BVALIDST</b>	Output	AXI3 device	Response valid
<b>BREADYST</b>	Input		Response ready
<b>BIDST[n]<sup>a</sup></b>	Output		Response ID
<b>BRESPST[1:0]</b>	Output		Write response

a. You can define the number of AXI ID bits on this port using the AXIST\_ID\_BIT build parameter.

## A.9 Performance monitoring signals

Table A-32 shows the performance monitoring signals.

**Table A-32 Performance monitoring signals**

Name	Type	Source/destination	Description
<b>PMUEVENT0[55:0]</b>	Output	<i>Performance Monitoring Unit (PMU)</i> or External Performance Monitoring Unit	PMU event bus for processor 0.
<b>PMUEVENT1[55:0]</b>	Output		PMU event bus for processor 1.
<b>PMUIRQ[N:0]</b>	Output	System Integrity Controller or External Performance Monitoring unit	Processor 0 and processor 1 interrupt requests by system metrics.
<b>PMUPRIV[N:0]</b>	Output	External Performance Monitoring Unit	<p>Gives the status of the Cortex-R7 MPCore processor:</p> <p>0            In user mode.</p> <p>1            In privileged mode.</p> <p>———— <b>Note</b> ————</p> <p>This signal does not provide input to the CoreSight trace delivery infrastructure.</p>

See *Performance Monitoring Unit* on page 10-3.

## A.10 Exception flag signals

Table A-33 shows the exception flag signals.

**Table A-33 Exception flag signals**

Name	Type	Source/destination	Description
<b>SCUEVABORT</b>	Output	System integrity controller	Indicates that an external abort has occurred during a coherency writeback. It is a pulse signal that is asserted for one <b>CLK</b> clock cycle.
<b>FPUFLAGS0[5:0]</b>	Output		Floating-Point Unit output flags for processor 0. Only implemented if processor 0 includes an FPU: Bit[5] gives the value of FPSCR[7]. Bits[4:0] give the value of FPSCR[4:0].
<b>FPUFLAGS1[5:0]</b>	Output		Floating-Point Unit output flags for processor 1. Only implemented if processor 1 is present and if processor 1 includes an FPU: Bit[5] gives the value of FPSCR[7]. Bits[4:0] give the value of FPSCR[4:0].

See Chapter 5 *Floating Point Unit Programmers Model*.

## A.11 Error detection notification signals

This section describes the following error detection notification signals:

- [Error detection global notification signals.](#)
- [RAM ECC error bank status signals.](#)
- [Bus ECC error signals on AXI master ports on page A-26.](#)
- [Bus ECC error signals on AXI peripheral ports on page A-27.](#)
- [Bus ECC error signals on AXI slave ports on page A-28.](#)
- [Lock-step and split/lock signals on page A-30.](#)

### A.11.1 Error detection global notification signals

[Table A-34](#) shows the error detection global notification signals.

**Table A-34 Error detection global notification signals**

Name	Type	Source/destination	Description
<b>RAMERR</b>	Output	Processor 0 and processor 1 RAM arrays and SCU RAMs	Any ECC error on any RAM
<b>FATALRAMERR[N:0]</b>	Output		Fatal ECC error on any RAM
<b>ITCMECCEN</b>	Input		Defines reset value of ACTLR bit[10] for each processor
<b>CORRBUSERR</b>	Output	SCU	Correctable ECC error on any bus
<b>FATALBUSERR</b>	Output		Fatal ECC error on any bus

### A.11.2 RAM ECC error bank status signals

[Table A-35](#) shows the RAM ECC error bank status signals.

**Table A-35 RAM ECC error bank status signals**

Name	Type	Source/destination	Description
<b>DCEBEMPTY[N:0]</b>	Output	Specific RAM group	ECC error bank empty for data cache
<b>ICEBEMPTY[N:0]</b>	Output		ECC error bank empty for instruction cache
<b>DTCMEBEMPTY0</b>	Output		ECC error bank empty for data TCM for processor 0
<b>DTCMEBEMPTY1</b>	Output		ECC error bank empty for data TCM for processor 1
<b>ITCMEBEMPTY0</b>	Output		ECC error bank empty for instruction TCM for processor 0
<b>ITCMEBEMPTY1</b>	Output		ECC error bank empty for instruction TCM for processor 1
<b>SCUEBEMPTY</b>	Output		ECC error bank empty for SCU

### A.11.3 Bus ECC error signals on AXI master ports

Table A-36 shows the bus ECC error signals on the AXI master ports. These signals are only present if ECC is implemented. **x** in the signal name represents either AXI Master Port 0 or AXI Master Port 1.

**Table A-36 Bus ECC error signals on AXI master ports**

Name	Type	Source/destination	Description
<b>AXICORRERRM<sub>x</sub></b>	Output	AXI master port	Correctable error on DR channel of AXI master port.
<b>AXIFATALERRM<sub>x</sub>[4:0]</b>	Output		Fatal error on AXI master port: [4] fatal error on DR channel. [3] fatal error on AR channel. [2] fatal error on DB channel. [1] fatal error on DW channel. [0] fatal error on AW channel.
<b>ARVALIDPTYM<sub>x</sub></b>	Output		Parity for address valid.
<b>ARREADYPTYM<sub>x</sub></b>	Input		Parity for address ready.
<b>ARADDRPTYM<sub>x</sub>[3:0]</b>	Output		Parity for address.
<b>ARCTLPTYM<sub>x</sub>[3:0]</b>	Output		Parity signals: [0] parity for address ID. [1] parity for burst length. [2] parity for burst size, burst type, and lock type. [3] parity for cache type and protection.
<b>ARUSERPTYM<sub>x</sub></b>	Output		Parity for transfer attributes.
<b>RVALIDPTYM<sub>x</sub></b>	Input		Parity for read valid.
<b>RREADYPTYM<sub>x</sub></b>	Output		Parity for read ready.
<b>RCTLPTYM<sub>x</sub>[1:0]</b>	Input		Parity signals: [0] parity for read ID. [1] parity for read response and read last.
<b>RDATAERRCODEM<sub>x</sub>[7:0]</b>	Input		ECC bits on data bus, when BUS_ECC build parameter is set.
<b>AWVALIDPTYM<sub>x</sub></b>	Output		Parity for address valid.
<b>AWREADYPTYM<sub>x</sub></b>	Input		Parity for address ready.
<b>AWADDRPTYM<sub>x</sub>[3:0]</b>	Output		Parity for address.
<b>AWCTLPTYM<sub>x</sub>[3:0]</b>	Output		Parity signals: [0] parity for address ID. [1] parity for burst length. [2] parity for burst size, burst type, and lock type. [3] parity for cache type and protection.
<b>AWUSERPTYM<sub>x</sub></b>	Output		Parity for transfer attributes.
<b>WVALIDPTYM<sub>x</sub></b>	Output		Parity for write valid.
<b>WREADYPTYM<sub>x</sub></b>	Input		Parity for write ready.

Table A-36 Bus ECC error signals on AXI master ports (continued)

Name	Type	Source/destination	Description
WCTLPTYMx[2:0]	Output	AXI master port	Parity signals: [0] parity for write ID. [1] parity for write strobes. [2] parity for write last.
WUSERPTYMx	Output		Parity for transfer attributes.
WDATAERRCODEMx[7:0]	Output		ECC bits on data bus, when BUS_ECC build parameter is set.
BVALIDPTYMx	Input		Parity for response valid.
BREADYPTYMx	Output		Parity for response ready.
BCTLPTYMx[1:0]	Input		Parity signals: [0] parity for response ID. [1] parity for write response.

#### A.11.4 Bus ECC error signals on AXI peripheral ports

Table A-37 shows the bus ECC error signals on the AXI peripheral ports. These signals are only present if ECC is implemented.

Table A-37 Bus ECC error signals on AXI peripheral ports

Name	Type	Source/destination	Description
AXICORRERRMP	Output	AXI master port	Correctable error on DR channel of AXI peripheral port.
AXIFATALERRMP[4:0]	Output		Fatal error on AXI master port: [4] fatal error on DR channel. [3] fatal error on AR channel. [2] fatal error on DB channel. [1] fatal error on DW channel. [0] fatal error on AW channel.
ARVALIDPTYMP	Output		Parity for address valid.
ARREADYPTYMP	Input		Parity for address ready.
ARADDRPTYMP[3:0]	Output		Parity for address.
ARCTLPTYMP[3:0]	Output		Parity signals: [0] parity for address ID. [1] parity for burst length. [2] parity for burst size, burst type, and lock type. [3] parity for cache type and protection.
ARUSERPTYMP	Output		Parity for transfer attributes.
RVALIDPTYMP	Input		Parity for read valid.
RREADYPTYMP	Output		Parity for read ready.
RCTLPTYMP[1:0]	Input		Parity signals: [0] parity for read ID. [1] parity for read response.

Table A-37 Bus ECC error signals on AXI peripheral ports (continued)

Name	Type	Source/destination	Description
<b>RDATAERRCODEMP[6:0]</b>	Input	AXI master port	ECC bits on data bus, when BUS_ECC build parameter is set.
<b>AWVALIDPTYMP</b>	Output		Parity for address valid.
<b>AWREADYPTYMP</b>	Input		Parity for address ready.
<b>AWADDRPTYMP[3:0]</b>	Output		Parity for address.
<b>AWCTLPTYMP[3:0]</b>	Output		Parity signals: [0] parity for address ID. [1] parity for burst length. [2] parity for burst size, burst type, and lock type. [3] parity for cache type and protection.
<b>AWUSERPTYMP</b>	Output		Parity for transfer attributes.
<b>WVALIDPTYMP</b>	Output		Parity for write valid.
<b>WREADYPTYMP</b>	Input		Parity for write ready.
<b>WCTLPTYMP[2:0]</b>	Output		Parity signals: [0] parity for write ID. [1] parity for write strobes. [2] parity for write last.
<b>WUSERPTYMP</b>	Output		Parity for transfer attributes.
<b>WDATAERRCODEMP[6:0]</b>	Output		ECC bits on data bus, when BUS_ECC build parameter is set.
<b>BVALIDPTYMP</b>	Input		Parity for response valid.
<b>BREADYPTYMP</b>	Output		Parity for response ready.
<b>BCTLPTYMP[1:0]</b>	Input		Parity signals: [0] parity for response ID. [1] parity for write response.

### A.11.5 Bus ECC error signals on AXI slave ports

Table A-38 shows the bus ECC error signals on the AXI slave ports. These signals are only present if ECC is implemented.

Table A-38 Bus ECC error signals on AXI slave ports

Name	Type	Source/destination	Description
<b>AXICORRERRSC</b>	Output	AXI ACP	Correctable error on DW channel of AXI ACP port
<b>AXIFATALERRSC[4:0]</b>	Output		Fatal error on AXI ACP port: [4] fatal error on DR channel. [3] fatal error on AR channel. [2] fatal error on DB channel. [1] fatal error on DW channel. [0] fatal error on AW channel.
<b>ARVALIDPTYSC</b>	Input		Parity for address valid.

Table A-38 Bus ECC error signals on AXI slave ports (continued)

Name	Type	Source/destination	Description
ARREADYPTYSC	Output	AXI ACP	Parity for address ready.
ARADDRPTYSC[3:0]	Input		Parity for address.
ARCTLPTYSC[3:0]	Input		Parity signals: [0] parity for address ID. [1] parity for burst length. [2] parity for burst size, burst type, and lock type. [3] parity for cache type and protection.
ARUSERPTYSC	Input		Parity for transfer attributes.
RVALIDPTYSC	Output		Parity for read valid.
RREADYPTYSC	Input		Parity for read ready.
RCTLPTYSC[1:0]	Output		Parity signals: [0] parity for read ID. [1] parity for read response.
RDATAERRCODESC[7:0]	Output		ECC bits on data bus, when BUS_ECC build parameter is set.
AWVALIDPTYSC	Input		Parity for address valid.
AWREADYPTYSC	Output		Parity for address ready.
AWADDRPTYSC[3:0]	Input		Parity for address.
AWCTLPTYSC[3:0]	Input		Parity signals: [0] parity for address ID. [1] parity for burst length. [2] parity for burst size, burst type, and lock type. [3] parity for cache type and protection.
AWUSERPTYSC	Input		Parity for transfer attributes.
WVALIDPTYSC	Input		Parity for write valid.
WREADYPTYSC	Output		Parity for write ready.
WCTLPTYSC[2:0]	Input		Parity signals: [0] parity for write ID. [1] parity for write strobes. [2] parity for write last.
WDATAERRCODESC[7:0]	Input		ECC bits on data bus, when BUS_ECC build parameter is set.
BVALIDPTYSC	Output		Parity for response valid.
BREADYPTYSC	Input		Parity for response ready.
BCTLPTYSC[1:0]	Output		Parity signals: [0] parity for response ID. [1] parity for write response.



### A.11.6 Lock-step and split/lock signals

Table A-39 shows the lock-step and split/lock signals.

**Table A-39 Lock-step and split/lock signals**

Name	Type	Source/destination	Description
COMPENABLE	Input	System configuration	Enables comparison logic that compares the outputs of the processor, SCU, and AXI TCM slave with those of their redundant copy. See <a href="#">Redundant processor comparison on page 1-8</a> .
COMPFAULT	Output		Indicates a fault in either the main or redundant logic. See <a href="#">Redundant processor comparison on page 1-8</a> .
SAFEMODE	Input		Selects split mode or locked mode. See <a href="#">Split/lock on page 1-8</a> .

## A.12 Test interface

Table A-40 shows the test interface signals.

**Table A-40 Test interface signals**

Name	Type	Source/destination	Description
<b>DFTSE</b>	Input	External test interface	Scan shift enable
<b>DFTRAMHOLD</b>	Input		Holds RAM content during scan shift
<b>DFTRAMCLKENABLE</b>	Input		Forces RAM clock for DFT purposes even when processors are in WFI mode
<b>DFTTESTMODE</b>	Input		Disable/bypass logic for test purposes

## A.13 MBIST interface

The width of the MBIST interface signals varies depending on whether ECC is implemented or not. In addition, DTCM can be run at speed and is accessed through the same port as L1 RAM. ITCM has its own dedicated interface.

Table A-41 shows the L1 and DTCM signals for designs without ECC.

**Table A-41 L1 and DTCM Cortex-R7 MBIST interface width without ECC**

Name	Type	Source/destination	Description
MBISTACK1[1:0]	Output	MBIST controller	BIST mode acknowledge signal: 0 Enabled. 1 Not enabled.
MBISTREQ1[1:0]	Input		BIST mode request signal, one per processor.
MBISTCFG1[1:0]	Input		MBIST configuration.
MBISTADDR1[13:0]	Input		MBIST address.
MBISTINDATA1[63:0]	Input		MBIST data in.
MBISTOUTDATA1[63:0]	Output		MBIST data out.
MBISTWRITEEN1	Input		Global write enable.
MBISTREADEN1	Input		Global read enable.
MBISTARRAY1[5:0]	Input		MBIST arrays used for testing RAMs.
MBISTBE1[63:0]	Input		MBIST bit-write enable.

Table A-42 shows the L1 and DTCM signals for designs with ECC.

**Table A-42 L1 and DTCM Cortex-R7 MBIST interface width with ECC**

Name	Type	Source/destination	Description
MBISTACK1[1:0]	Output	MBIST controller	BIST mode acknowledge signal: 0 Enabled. 1 Not enabled.
MBISTREQ1[1:0]	Input		BIST mode request signal, one per processor.
MBISTCFG1[1:0]	Input		MBIST configuration.
MBISTADDR1[13:0]	Input		MBIST address.
MBISTINDATA1[77:0]	Input		MBIST data in.
MBISTOUTDATA1[77:0]	Output		MBIST data out.
MBISTWRITEEN1	Input		Global write enable.
MBISTREADEN1	Input		Global read enable.
MBISTARRAY1[5:0]	Input		MBIST arrays used for testing RAMs.
MBISTBE1[77:0]	Input		MBIST bit-write enable.

Table A-43 shows the ITCM signals for designs without ECC.

**Table A-43 ITCM Cortex-R7 MBIST interface width without ECC**

Name	Type	Source/destination	Description
MBISTACK2[1:0]	Output	MBIST controller	BIST mode acknowledge signal: 0 Enabled. 1 Not enabled.
MBISTREQ2[1:0]	Input		BIST mode request signal.
MBISTCFG2[1:0]	Input		MBIST all mode support.
MBISTADDR2[11:0]	Input		MBIST address.
MBISTINDATA2[63:0]	Input		MBIST data in.
MBISTOUTDATA2[63:0]	Output		MBIST data out.
MBISTWRITEEN2	Input		Global write enable.
MBISTREADEN2	Input		Global read enable.
MBISTARRAY2[4:0]	Input		MBIST arrays used for testing RAMs.
MBISTBE2	Input		MBIST bit-write enable.

Table A-44 shows the ITCM signals for designs with ECC.

**Table A-44 ITCM Cortex-R7 MBIST interface width with ECC**

Name	Type	Source/destination	Description
MBISTACK2[1:0]	Output	MBIST controller	BIST mode acknowledge signal: 0 Enabled. 1 Not enabled.
MBISTREQ2[1:0]	Input		BIST mode request signal.
MBISTCFG2[1:0]	Input		MBIST all mode support.
MBISTADDR2[11:0]	Input		MBIST address.
MBISTINDATA2[71:0]	Input		MBIST data in.
MBISTOUTDATA2[71:0]	Output		MBIST data out.
MBISTWRITEEN2	Input		Global write enable.
MBISTREADEN2	Input		Global read enable.
MBISTARRAY2[4:0]	Input		MBIST arrays used for testing RAMs.
MBISTBE2	Input		MBIST bit-write enable.

## A.14 External debug signals

Table A-45 shows the debug enable signals.

**Table A-45 Debug enable signals**

Name	Type	Source/destination	Description
<b>DBGEN</b> [N:0]	Input	External debug device	Individual processor invasive debug enable:
			0 Not enabled. 1 Enabled.
<b>NIDEN</b> [N:0]	Input		Individual processor noninvasive debug enable:
			0 Not enabled. 1 Enabled.

Table A-46 shows the debug signals.

**Table A-46 Debug signals**

Name	Type	Source/destination	Description
<b>EDBGRQ</b> [N:0]	Input	External debug device	Individual processor external debug request: 0 No external debug request 1 External debug request. The processor treats the <b>EDBGRQ</b> input as level sensitive. The <b>EDBGRQ</b> input must be asserted until the processor asserts <b>DBGACK</b> .
<b>DBGACK</b> [N:0]	Output		Individual processor debug acknowledge signal. Acknowledges that the corresponding processor has entered debug state after an external debug request.
<b>DBGCPUDONE</b> [N:0]	Output		Acknowledges that corresponding processor has entered debug state and that all previous non-debug state memory accesses are complete.
<b>DBGRESTART</b> [N:0]	Input		Individual processor signal that causes the processor to exit debug state. It must be held HIGH until <b>DBGRESTARTED</b> is deasserted: 0 Not enabled. 1 Enabled.
<b>DBGRESTARTED</b> [N:0]	Output		Individual processor signal. Used with <b>DBGRESTART</b> to move from debug state to normal state: 0 Not enabled. 1 Enabled.
<b>DBGNOPWRDWN</b> [N:0]	Output		Output reflecting the value of <b>DBGPRCR</b> [0]. See the <i>ARM Architecture Reference Manual</i> .
<b>DBGSWENABLE</b> [N:0]	Input		When LOW only the external debug agent can modify debug registers: 0 Not enabled. 1 Enabled. Access by the software through the extended CP14 interface is permitted. External CP14 and external debug accesses are permitted.

Table A-46 Debug signals (continued)

Name	Type	Source/destination	Description
<b>DBGROMADDR[31:12]</b>	Input	External debug device	CoreSight System configuration. Specifies bits[31:12] of the ROM table physical address. If the address cannot be determined, tie this signal off to zero.
<b>DBGROMADDRV</b>	Input		Valid signal for <b>DBGROMADDR</b> . If the address cannot be determined, tie this signal LOW.
<b>DBGSELFADDR[31:17]</b>	Input		Specifies bits[31:17] of the two's complement signed offset from the ROM Table physical address to the physical address where the debug registers are memory-mapped. If the offset cannot be determined, tie this signal off to zero.
<b>DBGSELFADDRV</b>	Input		Valid signal for <b>DBGSELFADDR</b> . If the offset cannot be determined, tie this signal LOW.

Table A-47 shows the miscellaneous debug signals.

Table A-47 Miscellaneous debug signals

Name	Type	Source/destination	Description
<b>COMMRX[N:0]</b>	Output	External debug device	Individual processor signal. Comms Channels Receive portion of Data Transfer Register full flag: 0 Empty. 1 Full.
<b>COMMTX[N:0]</b>	Output		Individual processor signal. Comms Channels Transmit portion of Data Transfer Register full flag: 0 Empty. 1 Full.

Table A-48 shows the Debug APB interface signals.

Table A-48 Debug APB interface signals

Name	Type	Source/destination	Description
<b>PENABLEDBG</b>	Input	CoreSight APB devices	APB clock enable. Indicates a second and subsequent cycle of a transfer.
<b>PRDATADBG[31:0]</b>	Output		APB read data bus.
<b>PSELDBG</b>	Input		Debug registers select: 0 Debug registers not selected. 1 Debug registers selected.
<b>PSLVERRDBG</b>	Output		APB slave error signal: 0 No transfer error. 1 Transfer error.
<b>PWRITEDBG</b>	Input		APB read/write signal.

Table A-48 Debug APB interface signals (continued)

Name	Type	Source/destination	Description
<b>PADDRDBG[16:2]</b>	Input	CoreSight APB devices	Programming address. Bits[16:12] have the following meaning: 00000 ROM table. 10000 Processor 0 debug. 10001 Processor 0 PMU. 10010 Processor 1 debug, if processor 1 is present, otherwise reserved. 10011 Processor 1 PMU, if processor 1 is present, otherwise reserved. 11000 CTI0 11001 CTI1, if processor 1 is present, otherwise reserved. 11100 ETM0 11101 ETM1, if ETM1 is present, otherwise reserved.
<b>PADDRDBG31</b>	Input		APB address bus bit[31]: 0 Not an external debugger access. 1 External debugger access.
<b>PREADYDBG</b>	Output		APB slave ready. An APB slave can assert <b>PREADY</b> to extend a transfer.
<b>PWDATADB[31:0]</b>	Input		APB write data.

## A.15 ETM/ATB interface signals

Table A-49 shows the ETM/ATB interface signals for data trace. **x** in the signal name represents either ETM0 or ETM1.

**Table A-49 ETM/ATB interface signals for data trace**

Signal name	Type	Source/destination	Description
AFREADYMDx	Output	Trace device	ATB interface FIFO flush finished
AFVALIDMDx	Input		ATB interface FIFO flush request
ATBYTESMDx[2:0]	Output		Size of <b>ATDATA</b>
ATDATAMDx[63:0]	Output		ATB interface data
ATIDMDx[6:0]	Output		ATB interface trace source ID
ATREADYMDx	Input		<b>ATDATA</b> can be accepted
ATVALIDMDx	Output		ATB interface data valid
SYNCREQDx	Input		Synchronization request from data trace sink

Table A-50 shows the ETM/ATB interface signals for instruction trace. **x** in the signal name represents either ETM0 or ETM1.

**Table A-50 ETM/ATB interface signals for instruction trace**

Signal name	Type	Source/destination	Description
AFREADYMIx	Output	Trace device	ATB interface FIFO flush finished
AFVALIDMIx	Input		ATB interface FIFO flush request
ATBYTESMIx[1:0]	Output		Size of <b>ATDATA</b>
ATDATAMIx[31:0]	Output		ATB interface data
ATIDMIx[6:0]	Output		ATB interface trace source ID
ATREADYMIx	Input		<b>ATDATA</b> can be accepted
ATVALIDMIx	Output		ATB interface data valid
SYNCREQIx	Input		Synchronization request from instruction trace sink

Table A-51 shows the miscellaneous trace signals. **x** in the signal name represents either ETM0 or ETM1.

**Table A-51 Miscellaneous trace signals**

Signal name	Type	Source/destination	Description
ETMACTIVEx	Output	Trace device	Trace is being output.
ETMIFENx	Output		Power control for ATB/ETM interface.
ETMPWRUPREQx	Output	System power control	Request to maintain power to ETM.
TSSIZE	Input	Tie off	When HIGH (1), timestamp is 64 bit. When LOW (0), timestamp is 48 bit.
TSVALUE[63:0]	Input	Trace device	Timestamp value.



Table A-52 shows the CTI signals.

**Table A-52 CTI signals**

Signal name	Type	Source/destination	Description
<b>CTICHIN[3:0]</b>	Input	Trace device	Channel in
<b>CTICHOUTACK[3:0]</b>	Input		Channel out acknowledge
<b>CTICHOUT[3:0]</b>	Output		Channel out
<b>CTICHINACK[3:0]</b>	Output		Channel in acknowledge
<b>CIHSBYPASS[3:0]</b>	Input		Channel interface HS bypass
<b>nCTHRQ[N:0]</b>	Output		Active-LOW interrupt from CTI

## A.16 Memory reconstruction port signals

Table A-53 shows the MRP signals. **x** in the signal name represents either processor 0 or processor 1.

**Table A-53 Memory reconstruction port signals**

Name	Type	Source/destination	Description
<b>MRPREADY<sub>x</sub></b>	Input	Trace analysis engine	Ready signal of any write access
<b>MRPVALID<sub>x</sub></b>	Output		Valid signal of any write access
<b>MRPADDR<sub>x</sub>[31:0]</b>	Output		Address of any write access
<b>MRPDAT<sub>x</sub>[63:0]</b>	Output		Data of any write access
<b>MRPSTRB<sub>x</sub>[7:0]</b>	Output		Strobe of any write access

## A.17 Power gating interface signals

Table A-54 shows the power gating interface signals. x in the signal name represents either processor 0 or processor 1.

**Table A-54 Power gating interface signals**

Name	Type	Source/destination	Description
<b>nPWRUPSCURAM</b>	Input	Power controller	SCU power switch enable
<b>nPWRUPACKSCURAM</b>	Output		SCU power switch acknowledge
<b>nPWRUPCPUx</b>	Input		Individual processor power switch enable
<b>nPWRUPACKCPUx</b>	Output		Individual processor power switch acknowledge
<b>nPWRUPCPUDRAMx</b>	Input		Individual processor data RAM power switch enable
<b>nPWRUPACKCPUDRAMx</b>	Output		Individual processor data RAM power switch acknowledge
<b>nPWRUPCPUIRAMx</b>	Input		Individual processor instruction RAM power switch enable
<b>nPWRUPACKCPUIRAMx</b>	Output		Individual processor instruction RAM power switch acknowledge
<b>nPWRUPDTCMx</b>	Input		Individual DTCM RAM power switch enable
<b>nPWRUPACKDTCMx</b>	Output		Individual TCM RAM power switch acknowledge
<b>nPWRUPITCMx</b>	Input		Individual ITCM RAM power switch enable
<b>nPWRUPACKITCMx</b>	Output		Individual TCM RAM power switch acknowledge
<b>nPWRUPDBG</b>	Input		Debug power switch enable
<b>nPWRUPACKDBG</b>	Output		Debug power switch acknowledge
<b>nPWRUPETMx</b>	Input		Individual ETM power switch enable
<b>nPWRUPACKETMx</b>	Output		Individual ETM power switch acknowledge
<b>nISOLATESCURAM</b>	Input		SCU RAM clamp control
<b>nISOLATECPUx</b>	Input		Individual processor clamp control
<b>nISOLATECPUDRAMx</b>	Input		Individual processor data RAM clamp control
<b>nISOLATECPUIRAMx</b>	Input		Individual processor instruction RAM clamp control
<b>nISOLATEDTCMx</b>	Input		Individual DTCM RAM clamp control
<b>nISOLATEITCMx</b>	Input		Individual ITCM RAM clamp control
<b>nISOLATEDBG</b>	Input		Debug clamp control
<b>nISOLATEETMx</b>	Input		Individual ETM clamp control

# Appendix B

## Cycle Timings and Interlock Behavior

This appendix describes the cycle timings of integer instructions on Cortex-R7 MPCore processors. It contains the following sections:

- *About instruction cycle timing on page B-2.*
- *Data-processing instructions on page B-3.*
- *Load and store instructions on page B-4.*
- *Multiplication instructions on page B-7.*
- *Branch instructions on page B-8.*
- *Serializing instructions on page B-9.*

## B.1 About instruction cycle timing

This appendix provides information to estimate how much execution time particular code sequences require. The complexity of the Cortex-R7 processor makes it impossible to calculate precise timing information manually. The timing of an instruction is often affected by other concurrent instructions, memory system activity, and additional events outside the instruction flow. Describing all possible instruction interactions, and all possible events that take place in the processor, is beyond the scope of this document.

## B.2 Data-processing instructions

Table B-1 shows the execution unit cycle time for data-processing instructions.

Table B-1 shows the following cases:

**no shift on source registers**

For example, ADD r0, r1, r2

**shift by immediate source register**

For example, ADD r0, r1, r2 LSL #2

**shift by register**

For example, ADD r0, r1, r2 LSL r3.

**Table B-1 Data-processing instructions cycle timings**

Instruction	No shift	Shift by	
		Constant	Register
MOV	1	1	2
AND, EOR, SUB, RSB, ADD, ADC, SBC, RSC, CMN, ORR, BIC, MVN, TST, TEQ, CMP	1	2	3
QADD, QSUB, QADD8, QADD16, QSUB8, QSUB16, SHADD8, SHADD16, SHSUB8, SHSUB16, UQADD8, UQADD16, UQSUB8, UQSUB16, UHADD8, UHADD16, UHSUB8, UHSUB16, QASX, QSAX, SHASX, SHSAX, UQASX, UQSAX, UHASX, UHSAX	2	-	-
QDADD, QDSUB, SSAT, USAT	3	-	-
PKHBT, PKHTB	1	2	-
SSAT16, USAT16, SADD8, SADD16, SSUB8, SSUB16, UADD8, UADD16, USUB8, USUB16, SASX, SSAX, UASX, USAX	1	-	-
SXTAB, SXTAB16, SXTAH, UXTAB, UXTAB16, UXTAH	3	-	-
SXTB, STXB16, SXTH, UXTB, UTXB16, UXTH	2	-	-
BFC, BFI, UBFX, SBFX	2	-	-
CLZ, MOVT, MOVW, RBIT, REV, REV16, REVSH, MRS	1	-	-
SDIV, UDIV	4-16	-	-
MSR not modifying mode or control bits. See <a href="#">Serializing instructions on page B-9</a> .	1	-	-

## B.3 Load and store instructions

Load and store instructions are classed as:

- Single load and store instructions such as LDR instructions.
- Load and store multiple instructions such as LDM instructions.

For load multiple and store multiple instructions, the number of registers in the register list usually determines the number of cycles required to execute a load or store instruction.

The Cortex-R7 MPCore processor has an optimized path from a load instruction to a subsequent data processing instruction, saving 1 cycle on the load-use penalty.

This path is used when the following conditions are met:

- The data-processing instruction is an arithmetical, a logical or a saturation operation.
- The data-processing instruction does not require any shift.
- The load instruction does not require sign extension.
- The load instruction is not conditional.

Table B-2 shows cycle timing for single load and store operations. The result latency is the latency of the first loaded register.

**Table B-2 Single load and store operation cycle timings**

Instruction cycles	AGU cycles	Result latency	
		Fast forward cases	Other cases
LDR ,[reg]	1	2	3
LDR ,[reg imm]			
LDR ,[reg reg]			
LDR ,[reg reg LSL #2]			
LDR ,[reg reg LSL #3]			
LDR ,[reg reg LSL reg]	2	3	4
LDR ,[reg reg LSR reg]			
LDR ,[reg reg ASR reg]			
LDR ,[reg reg ROR reg]			
LDR ,[reg reg, RRX]			
LDRB ,[reg]	1	2	3
LDRB ,[reg imm]			
LDRB ,[reg reg]			
LDRB ,[reg reg LSL #2]			
LDRB ,[reg reg LSL #3]			
LDRH ,[reg]			
LDRH ,[reg imm]			
LDRH ,[reg reg]			
LDRH ,[reg reg LSL #2]			
LDRH ,[reg reg LSL #3]			

**Table B-2 Single load and store operation cycle timings (continued)**

Instruction cycles	AGU cycles	Result latency	
		Fast forward cases	Other cases
LDRB ,[reg reg LSL reg]	2	3	4
LDRB ,[reg reg ASR reg]			
LDRB ,[reg reg LSL reg]			
LDRB ,[reg reg ASR reg]			
LDRH ,[reg reg LSL reg]			
LDRH ,[reg reg ASR reg]			
LDRH ,[reg reg LSL reg]			
LDRH ,[reg reg ASR reg]			

The Cortex-R7 MPCore processor can load or store two 32-bit registers in each cycle. However, to access 64 bits, the address must be 64-bit aligned.

This scheduling is done in the *Address Generation Unit* (AGU). The number of cycles required by the AGU to process the load multiple or store multiple operations depends on the length of the register list and the 64-bit alignment of the address. The resulting latency is the latency of the first loaded register. [Table B-3](#) shows the cycle timings for load multiple operations.

**Table B-3 Load multiple operations cycle timings**

Instruction	AGU cycles to process the instruction		Resulting latency	
	Address aligned on a 64-bit boundary		Fast forward case	Other cases
	Yes	No		
LDM ,{1 register}	1	1	2	3
LDM ,{2 registers}	1	2	2	3
LDRD				
RFE				
LDM ,{3 registers}	2	2	2	3
LDM ,{4 registers}	2	3	2	3
LDM ,{5 registers}	3	3	2	3
LDM ,{6 registers}	3	4	2	3
LDM ,{7 registers}	4	4	2	3
LDM ,{8 registers}	4	5	2	3
LDM ,{9 registers}	5	5	2	3
LDM ,{10 registers}	5	6	2	3
LDM ,{11 registers}	6	6	2	3
LDM ,{12 registers}	6	7	2	3
LDM ,{13 registers}	7	7	2	3



Table B-3 Load multiple operations cycle timings (continued)

Instruction	AGU cycles to process the instruction		Resulting latency	
	Address aligned on a 64-bit boundary		Fast forward case	Other cases
	Yes	No		
LDM ,{14 registers}	7	8	2	3
LDM ,{15 registers}	8	8	2	3
LDM ,{16 registers}	8	9	2	3

Table B-4 shows the cycle timings of store multiple operations.

Table B-4 Store multiple operations cycle timings

Instruction	AGU cycles	
	Aligned on a 64-bit boundary	
	Yes	No
STM ,{1 register}	1	1
STM ,{2 registers}	1	2
STRD SRS		
STM ,{3 registers}	2	2
STM ,{4 registers}	2	3
STM ,{5 registers}	3	3
STM ,{6 registers}	3	4
STM ,{7 registers}	4	4
STM ,{8 registers}	4	5
STM ,{9 registers}	5	5
STM ,{10 registers}	5	6
STM ,{11 registers}	6	6
STM ,{12 registers}	6	7
STM ,{13 registers}	7	7
STM ,{14 registers}	7	8
STM ,{15 registers}	8	8
STM ,{16 registers}	8	9

## B.4 Multiplication instructions

Table B-5 shows the cycle timings for multiplication instructions.

**Table B-5 Multiplication instruction cycle timings**

Instruction	Cycles	Result latency
MUL(S), MLA(S)	2	4
SMULL(S), UMULL(S), SMLAL(S), UMLAL(S)	3	4 for the first written register 5 for the second written register
SMULxy, SMLAxy, SMULWy, SMLAWy	1	3
SMLALxy	2	3 for the first written register 4 for the second written register
SMUAD, SMUADX, SMLAD, SMLADX, SMUSD, SMUSDx, SMLSD, SMLSDx	1	3
SMMUL, SMMULR, SMLLA, SMLLAR, SMMLS, SMMLSR	2	4
SMLALD, SMLALDX, SMLS LD, SMLDL DX	2	3 for the first written register 4 for the second written register
UMAAL	3	4 for the first written register 5 for the second written register

## B.5 Branch instructions

Branch instructions have different timing characteristics:

- Branch instructions to immediate locations do not consume execution unit cycles.
- Data-processing instructions to the PC register are processed in the execution units as standard instructions. See [Data-processing instructions on page B-3](#).
- Load instructions to the PC register are processed in the execution units as standard instructions. See [Load and store instructions on page B-4](#).

See [Branch prediction on page 8-11](#) for information on dynamic branch prediction.

## B.6 Serializing instructions

Out of order execution is not always possible. Some instructions are serializing. Serializing instructions force the processor to complete all modifications to flags and general-purpose registers by previous instructions before the next instruction is executed.

This section describes timings for serializing instructions.

### B.6.1 Serializing instructions

The following exception entry instructions are serializing:

- SVC.
- SMC.
- BKPT.
- Instructions that take the prefetch abort handler.
- Instructions that take the Undefined Instruction exception handler.

The following instructions that modify mode or program control are serializing:

- MSR CPSR when they modify control or mode bits.
- Data processing to PC with the S bit set (for example, `MOVS pc, r14`).
- `LDM pc ^`.
- CPS.
- SETEND.
- RFE.

The following instructions are serializing:

- All MCR to CP14 or CP15 except ISB and DMB.
- MRC p14 for debug registers.
- WFE, WFI, SEV.
- CLREX.
- DSB.

The following instruction, that modifies the SPSR, is serializing:

- MSR SPSR.

# Appendix C

## Revisions

This appendix describes the technical changes between released issues of this book.

**Table C-1 Issue A**

Change	Location	Affects
First release	-	-

**Table C-2 Differences between issue A and issue B**

Change	Location	Affects
Updated the following terminology to conform to the <i>ARM Architecture Reference Manual</i> : <ul style="list-style-type: none"><li>• Precise abort changed to synchronous abort.</li><li>• Imprecise abort changed to asynchronous abort.</li><li>• Internal monitor changed to local monitor.</li><li>• External monitor changed to global monitor.</li></ul>	Throughout book.	All revisions
Added Debug and GIC architecture.	<a href="#">Compliance on page 1-3</a>	All revisions
Added CTM interface.	<a href="#">Interfaces on page 1-5</a>	All revisions
Updated block diagram.	<a href="#">Figure 2-1 on page 2-2</a>	All revisions
Updated PMU architecture version.	<a href="#">Debug and Trace on page 2-3</a>	All revisions
Updated description of <b>CLK</b> .	<a href="#">CLK on page 2-5</a>	All revisions

Table C-2 Differences between issue A and issue B (continued)

Change	Location	Affects
Updated description of WFI and WFE standby modes.	<a href="#">Standby modes on page 2-14</a> and <a href="#">Standby mode with RAM retention on page 2-15</a>	All revisions
Updated description of individual processor debug reset.	<a href="#">Individual processor debug reset on page 2-9</a>	All revisions
Added timing diagram for powerdown and powerup sequence.	<a href="#">Dormant mode on page 2-16</a>	All revisions
Updated description of AXI master port 1.	<a href="#">AXI master port 1 on page 2-19</a>	All revisions
Updated description of AXI peripheral port.	<a href="#">AXI peripheral port on page 2-20</a>	All revisions
Updated description of AXI TCM slave port.	<a href="#">AXI TCM slave port on page 2-21</a>	All revisions
Updated reset value for MIDR.	<a href="#">Table 4-2 on page 4-4</a> and <a href="#">Table 4-10 on page 4-11</a>	r0p1
Updated reset value for CTR.	<a href="#">Table 4-2 on page 4-4</a> , <a href="#">Table 4-10 on page 4-11</a> , <a href="#">Table 10-3 on page 10-4</a> , and <a href="#">Table 10-10 on page 10-17</a>	All revisions
Updated reset value for DTCMRR and ITCMRR.	<a href="#">Table 4-7 on page 4-8</a> and <a href="#">Table 4-14 on page 4-15</a>	All revisions
Updated description of CL 1 bits.	<a href="#">Table 4-23 on page 4-22</a>	All revisions
Updated description of bits[31,14].	<a href="#">Table 4-25 on page 4-24</a>	All revisions
Updated description of bits[31:30].	<a href="#">Table 4-27 on page 4-28</a>	All revisions
Deleted reference to Physical Address Register.	<a href="#">Chapter 4 System Control</a>	All revisions
Changed DTCMR to DTCMRR and ITCMR to ITCMRR.	<a href="#">Chapter 4 System Control</a>	All revisions
Separated DEER0-2 and IEER0-2 to show different bit assignment values.	<a href="#">Table 4-39 on page 4-40</a> and <a href="#">Table 4-40 on page 4-40</a>	All revisions
Updated description of bits[31:28].	<a href="#">Table 4-39 on page 4-40</a> and <a href="#">Table 4-40 on page 4-40</a>	All revisions
Updated usage of the Cache and TCM Debug Operation Register.	<a href="#">Using the CTDOR on page 4-42</a>	All revisions
Updated version and reset value of FPSID.	<a href="#">Table 5-2 on page 5-6</a> and <a href="#">Table 5-4 on page 5-8</a>	All revisions
Updated description of L1 memory system.	<a href="#">About the L1 memory system on page 6-2</a>	All revisions
Moved description of data cache policy from Introduction chapter.	<a href="#">Data cache policy on page 6-2</a>	All revisions
Updated description of external faults.	<a href="#">External faults on page 6-3</a>	All revisions
Updated cache disable and enable sequences.	<a href="#">Cache interaction with memory system on page 6-15</a>	All revisions
Updated description of memory types and their behavior.	<a href="#">Memory types and L1 memory system behavior on page 6-18</a>	All revisions
Updated bit allocations for Instruction tag RAM and SCU tag.	<a href="#">Table 7-1 on page 7-5</a>	All revisions
Updated RAM protection descriptions.	<a href="#">Table 7-1 on page 7-5</a>	All revisions
Updated RAM configuration values.	<a href="#">Table 7-3 on page 7-8</a>	All revisions
Deleted GPER and BPER from list of registers used in ECC.	<a href="#">Processor registers on page 7-12</a>	All revisions
Updated note about ECC on the ACP bus.	<a href="#">ECC on external AXI bus on page 7-7</a>	All revisions
Updated description of lock-step.	<a href="#">Lock-step on page 7-14</a>	All revisions

Table C-2 Differences between issue A and issue B (continued)

Change	Location	Affects
Updated description of split/lock.	<i>Static split/lock</i> on page 7-16	All revisions
Deleted default memory map table. This information is covered in the <i>ARM Architecture Reference Manual</i> .	<i>Memory Protection Unit</i> on page 8-3	All revisions
Updated description of low latency interrupt mode.	<i>Low latency interrupt mode</i> on page 8-12	All revisions
Updated description of SCU registers.	<i>SCU registers</i> on page 9-5	All revisions
Deleted reference to footnote for registers at offsets 0x70 and 0x74.	Table 9-2 on page 9-5	All revisions
Clarified description of interrupt controller clock frequency.	<i>Interrupt controller clock frequency</i> on page 9-19	All revisions
Clarified description of legacy <b>nFIQ</b> input.	<i>Interrupt distributor interrupt sources</i> on page 9-19	All revisions
Deleted reference to Secure accesses.	Table 9-18 on page 9-22	All revisions
Added text about UNPREDICTABLE accesses.	<i>Distributor register descriptions</i> on page 9-20 <i>Interrupt interface register descriptions</i> on page 9-27 <i>Private timer and watchdog registers</i> on page 9-29 <i>Global Timer Counter Registers</i> on page 9-35	All revisions
Updated reset value of ICCBPR.	Table 9-23 on page 9-27	All revisions
Updated description of <b>WDRESETREQ</b> .	<i>Watchdog Counter Register</i> on page 9-31	All revisions
Deleted reference to integration test registers.	Chapter 10 <i>Monitoring, Trace, and Debug</i>	All revisions
Deleted reference to CP14 interface access.	<i>PMU management registers</i> on page 10-4	All revisions
Updated description of processor events.	Table 10-7 on page 10-7	All revisions
Updated description of debug registers.	<i>Debug register descriptions</i> on page 10-15	All revisions
Removed reference to CP14 access to processor ID registers.	Table 10-8 on page 10-13	All revisions
Deleted description of breakpoint and watchpoint registers. This information is covered in the <i>ARM Architecture Reference Manual</i> .	<i>Breakpoint and Watchpoint Registers, DBGVn, DBGBCRn, DBGWVRn, and DBGWCRn</i> on page 10-16	All revisions
Updated access type of ITCTRL.	<i>Debug management registers</i> on page 10-16	All revisions
Updated description of processor ID registers.	<i>Processor ID Registers</i> on page 10-17	All revisions
Updated description of debug registers.	<i>Debug register descriptions</i> on page 10-15	All revisions
Added description of trigger inputs and outputs for CTI.	<i>Trigger inputs and outputs</i> on page 10-22	All revisions
Added AXI3 master interface attributes.	<i>About the L2 interface</i> on page 11-2	All revisions
Updated supported AXI3 transfers.	<i>Supported AXI3 transfers</i> on page 11-3	All revisions
Updated descriptions for the following: <ul style="list-style-type: none"> <li><b>ARUSERM0</b> and <b>ARUSERM1</b>.</li> <li><b>AWUSERM0</b> and <b>AWUSERM1</b>.</li> <li><b>ARUSERMP</b>.</li> <li><b>AWUSERMP</b>.</li> <li><b>WUSERM0</b> and <b>WUSERM1</b>.</li> </ul>	<i>AXI3 USER bits</i> on page 11-3	All revisions

Table C-2 Differences between issue A and issue B (continued)

Change	Location	Affects
Updated TCM sizes.	<a href="#">Accessing RAMs using the AXI3 interface on page 11-9</a>	All revisions
Updated all signals to match RTL.	<a href="#">Appendix A Signal Descriptions</a>	All revisions
Updated description of <b>nFIQOUT[N:0]</b> and <b>nIRQOUT[N:0]</b> .	<a href="#">Table A-3 on page A-6</a>	All revisions
Updated description of <b>SMPnAMP[N:0]</b> .	<a href="#">Table A-4 on page A-7</a>	All revisions
Updated description of <b>INCLKENMx</b> and <b>OUTCLKENMx</b> .	<a href="#">Table A-8 on page A-11</a>	All revisions
Updated description of <b>ARLENMx[3:0]</b> .	<a href="#">Table A-9 on page A-11</a>	All revisions
Updated description of <b>AWLENMx[3:0]</b> .	<a href="#">Table A-11 on page A-13</a>	All revisions
Updated descriptions of the following signals: <ul style="list-style-type: none"> <li>• <b>DBGACK[N:0]</b>.</li> <li>• <b>DBGCPUDONE[N:0]</b>.</li> <li>• <b>DBGRESTARTED[N:0]</b>.</li> <li>• <b>DBGNOPWRDWN[N:0]</b>.</li> </ul>	<a href="#">Table A-46 on page A-34</a>	All revisions
Added CTI signals.	<a href="#">Table A-52 on page A-38</a>	All revisions
Added power gating interface signals.	<a href="#">Table A-54 on page A-40</a>	All revisions
Added appendix for instruction cycle timing.	<a href="#">Appendix B Cycle Timings and Interlock Behavior</a>	All revisions

Table C-3 Differences between issue B and issue C

Change	Location	Affects
Updated the role of <b>COMPENABLE</b>	<a href="#">Redundant processor comparison on page 1-8</a>	All revisions
Updated the information about TCM RAM	<a href="#">Test features on page 1-9</a>	All revisions
Updated the description of the reset sequence	<a href="#">Cortex-R7 MPCore powerup reset on page 2-7</a>	All revisions
Updated the information about architectural registers after reset	<a href="#">Initialization on page 2-9</a>	All revisions
Updated description of exiting from WFE mode	<a href="#">Wait for Event on page 2-14</a>	All revisions
Added information about access from the DTCM	<a href="#">About the L1 memory system on page 6-2</a>	All revisions
Added information about multiple-bit errors	<a href="#">Protection method on page 7-3</a>	All revisions
Updated section	<a href="#">Low latency interrupt mode on page 8-12</a>	All revisions
Added caution about slave being private to the processor	<a href="#">System configurability and QoS on page 8-13</a>	All revisions
Updated the information about storage	<a href="#">Instruction and data TCM on page 8-15</a>	All revisions
Updated the information about <b>PFILTERSTART</b> and <b>PFILTEREND</b>	<a href="#">SCU registers on page 9-5</a>	All revisions
Updated the usage constraints	<a href="#">SCU CPU Power Status Register on page 9-9</a>	All revisions
Updated the reset value of the ICDIIDR and the range of subsequent registers	<a href="#">Table 9-17 on page 9-21</a>	All revisions



Table C-3 Differences between issue B and issue C (continued)

Change	Location	Affects
Corrected the register name from <b>ICDIPTRn</b> to <b>ICDIPRn</b>	<a href="#">Table 9-17 on page 9-21</a>	All revisions
Updated the information about the ICDIPRn and the ICCPMR	<ul style="list-style-type: none"> <li><a href="#">Table 9-17 on page 9-21.</a></li> <li><a href="#">Table 9-23 on page 9-27.</a></li> </ul>	All revisions
Updated the field values	<a href="#">Table 9-20 on page 9-24</a>	All revisions
Added information about comparator registers	<a href="#">Global timer on page 9-35</a>	All revisions
Updated the description of events 0x96-0x99	<a href="#">Table 10-7 on page 10-7</a>	All revisions
Updated information about read issuing capability	<a href="#">Table 11-1 on page 11-2</a>	All revisions
Updated section	<a href="#">ACP limitations on page 11-13</a>	All revisions
Updated the description of <b>ARLENMx</b>	<a href="#">Table A-9 on page A-11</a>	All revisions
Updated the information about <b>RDATAMP</b>	<a href="#">Table A-16 on page A-16</a>	All revisions
Updated description of <b>ACLKENS</b>	<a href="#">Table A-20 on page A-18</a>	All revisions
Updated description of <b>RCTLPTYMx</b>	<a href="#">Table A-36 on page A-26</a>	All revisions
Updated directionality of signals	<ul style="list-style-type: none"> <li><a href="#">Table A-22 on page A-19.</a></li> <li><a href="#">Table A-28 on page A-21.</a></li> <li><a href="#">Table A-49 on page A-37.</a></li> <li><a href="#">Table A-50 on page A-37.</a></li> <li><a href="#">Table A-51 on page A-37.</a></li> </ul>	All revisions
Updated width of <b>RDATAERRCODEM</b>	<a href="#">Table A-36 on page A-26</a>	All revisions
Updated width of <b>WDATAERRCODEMP</b>	<a href="#">Table A-37 on page A-27</a>	All revisions
Updated width of <b>RDATAERRCODESC</b>	<a href="#">Table A-38 on page A-28</a>	All revisions
Changed signal name from <b>AWREADPTYM0/M1/MP/SC</b> to <b>AWREADYPTYM0/M1/MP/SC</b>	<ul style="list-style-type: none"> <li><a href="#">Table A-36 on page A-26.</a></li> <li><a href="#">Table A-37 on page A-27.</a></li> <li><a href="#">Table A-38 on page A-28.</a></li> </ul>	All revisions