Arm® Cortex®-R8 MPCore Processor

Revision: r0p3

Technical Reference Manual

arm

100400_0003_01_en

Arm® Cortex®-R8 MPCore Processor

Technical Reference Manual

Release Information

Document History

Issue	Date	Confidentiality	Change
0000-01	14 December 2015	Confidential	First release for r0p0
0001-02	09 March 2016	Confidential	First release for r0p1
0001-03	03 March 2017	Non-Confidential	Second release for r0p1
0002-00	10 September 2018	Non-Confidential	First release for r0p2
0003-01	13 February 2019	Non-Confidential	First release for r0p3

Non-Confidential Proprietary Notice

110 Fulbourn Road, Cambridge, England CB1 9NJ.

LES-PRE-20349

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.

Unrestricted Access is an Arm internal classification.

Product Status

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

Web Address

http://www.arm.com

Contents
Arm® Cortex®-R8 MPCore Processor Technical Reference Manual

Preface

This preface introduces the *Arm® Cortex®-R8 MPCore Processor Technical Reference Manual*.

It contains the following:

About this book

Arm® Cortex®-R8 MPCore Technical Reference Manual (TRM) providing reference information on the processor design, implementation, registers, and interfaces. The guide includes documentation on the Memory Protection Unit (MPU), interrupt controller, debug, level 1 and level 2 interfaces. Available as PDF.

Product revision status

The r*m*p*n* identifier indicates the revision status of the product described in this book, for example, r*1*p*2*, where:

r*m* Identifies the major revision of the product, for example, r1.

p*n* Identifies the minor revision or modification status of the product, for example, p2.

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®-R8 processor.

Using this book

This book is organized into the following chapters:

Chapter 1 Introduction
This chapter introduces the Cortex-R8 processor and its features.

Chapter 2 Functional Description
This chapter describes the functionality of the Cortex-R8 processor.

Chapter 3 Programmers Model
This chapter describes the programmers model.

Chapter 4 System Control
This chapter describes the system control registers, their structure and operation, and how to use them.

Chapter 5 Floating Point Unit Programmers Model
This chapter describes the programmers model of the optional *Floating-Point Unit* (FPU).

Chapter 6 Level 1 Memory System
This chapter describes the L1 memory system.

Chapter 7 Fault Detection
This chapter describes the fault detection features of the Cortex-R8 processor.

Chapter 8 Determinism Support
This chapter describes the determinism support features of the Cortex-R8 processor.

Chapter 9 Multiprocessing
This chapter describes the multiprocessing features of the Cortex-R8 processor, including the SCU.

Chapter 10 Monitoring, Trace, and Debug
This chapter describes the monitoring, trace, and debug features of the Cortex-R8 processor.

Chapter 11 Embedded Trace Macrocell
This chapter describes the Embedded Trace Macrocell for the Cortex-R8 processor.

Chapter 12 Level 2 Interface
This chapter describes the L2 memory interface.

Appendix A Signal Descriptions

This appendix describes the Cortex-R8 processor signals. Signal name, direction, and source/destination details are provided for each signal.

Appendix B Cycle Timings and Interlock Behavior

This appendix describes the cycle timings of integer instructions on Cortex-R8 processor cores.

Appendix C Revisions

This appendix describes 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 the *Arm® Glossary* for more information.

Typographic conventions

italic

Introduces special terminology, denotes cross-references, and citations.

bold

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.

`monospace`

Denotes a permitted abbreviation for a command or option. You can enter the underlined text instead of the full command or option name.

`monospace italic`

Denotes arguments to monospace text where the argument is to be replaced by a specific value.

`monospace bold`

Denotes language keywords when used outside example code.

`<and>`

Encloses 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 *Arm® Glossary*. For example, IMPLEMENTATION DEFINED, IMPLEMENTATION SPECIFIC, UNKNOWN, and UNPREDICTABLE.

Timing diagrams

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

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

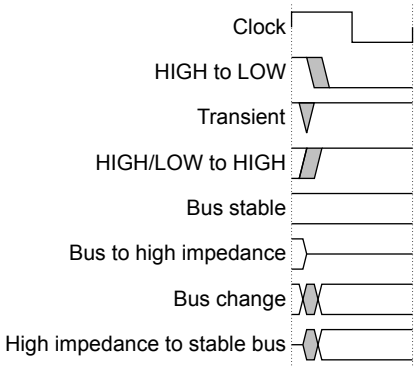


Figure 1 Key to timing diagram conventions

Signals

The signal conventions are:

Signal level

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

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

Lowercase n

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

Additional reading

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

Arm publications

- *Arm® Cortex®-R8 MPCore Integration Manual* (100402).
- *Arm® Cortex®-R8 MPCore Processor Configuration and Sign-off Guide* (100401).
- *Arm®Architecture Reference Manual Armv7-A and Armv7-R edition* (DDI 0406).
- *Arm® Embedded Trace Macrocell Architecture Specification ETMv4* (IHI 0064).
- *Arm® CoreSight™ Architecture Specification* (IHI 0029).
- *Arm® CoreSight™ SoC-400 Technical Reference Manual* (DDI 0480).
- *Arm® CoreSight™ SoC-400 Implementation Guide* (DII 0267).
- *Arm® CoreSight™ SoC-400 Integration Manual* (DIT 0037).
- *Arm® CoreSight™ SoC-400 System Design Guide* (DGI 0018).
- *Arm® CoreSight™ SoC-400 User Guide* (DUI 0563).
- *Arm® CoreSight™ DAP-Lite2 Technical Reference Manual* (DDI 0316).
- *Arm® AMBA® AXI and ACE Protocol Specification AXI3, AXI4, and AXI4-Lite, ACE and ACE-Lite* (IHI 0022).
- *Arm® AMBA® 3 AHB-Lite Protocol Specification* (IHI 0033).
- *Arm® AMBA® APB Protocol Specification* (IHI 0024).
- *Arm® Design Simulation Model User Guide* (DUI 0302).
- *Arm® Generic Interrupt Controller Architecture Specification* (IHI 0048B).
- *Arm® CoreLink™ Level 2 Cache Controller L2C-310 Technical Reference Manual* (DDI 0246).
- *Arm® Debug Interface Architecture Specification, ADIv5.0 to ADIv5.2* (IHI 0031).

Other publications

This section lists relevant documents published by third parties:

- None.

Feedback

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*. Give:

- The title *Arm Cortex-R8 MPCore Processor Technical Reference Manual*.
- The number 100400_0003_01_en.
- If applicable, the page number(s) to which your comments refer.
- 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-R8 processor and its features.

It contains the following sections:

1.1 About the Cortex®-R8 processor

The Cortex-R8 processor is a mid-range processor for use in deeply-embedded, real-time systems. 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 four cores. 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-R8 processor provides two specialized memory solutions for low-latency and determinism:

- *Tightly-Coupled Memory* (TCM) offers determinism and very low latency, but has limited memory space.
- Local RAM, cached by L1, offers a better 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) can be used on all 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

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

The Cortex-R8 processor complies with, or implements, the specifications that are described in the following sections.

1.2.1 Arm® architecture

The Cortex-R8 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 *Floating Point Unit* (FPU) implements the VFPv3-D16 architecture that includes the VFPv3 instruction set. See the *Arm® Architecture Reference Manual, Arm®v7-A and Arm®v7-R edition*.

1.2.2 Trace macrocell

Each Cortex-R8 core can include an optional *Embedded Trace Macrocell* (ETM) that can be shared with other cores, or each core can have an optional dedicated ETM. This ETM implements ETM architecture version 4.

See *Arm® Embedded Trace Macrocell Architecture Specification ETMv4*.

1.2.3 Advanced Microcontroller Bus Architecture

The Cortex-R8 processor complies with the AMBA 3 protocol.

See *Arm® AMBA® AXI and ACE Protocol Specification AXI3, AXI4, and AXI4-Lite, ACE and ACE-Lite* and *Arm® AMBA® 3 APB Protocol Specification*.

1.2.4 Debug architecture

The Cortex-R8 processor implements version 7 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-R8 processor implements the Arm *Generic Interrupt Controller* (GIC) v1.0 architecture.

See the *Arm® Generic Interrupt Controller Architecture Specification*.

1.3 Processor features

The Cortex-R8 processor includes up to four cores, a *Snoop Control Unit* (SCU), integrated timers, the ability to implement redundant processor logic, and numerous other advanced features.

The Cortex-R8 processor includes the following features:

- Up to four cores, with the following features:
  - A 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.
  - An optional *Floating Point Unit* (FPU) in each core. 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.
  - An APB Debug slave interface.
  - An optional *Memory Reconstruction Port* (MRP) for each core for memory reconstruction.
  - A *Performance Monitoring Unit* (PMU) in each core.
  - An ECC fatal error signal that indicates data written from the core might be corrupted. The system can use this signal to disable writes to external memory.
- A Harvard L1 memory system for each core with:
  - An Armv7-R architecture *Memory Protection Unit* (MPU) with 12, 16, 20 or 24 regions, each region with a minimum resolution of 256 bytes.
  - Optional data and instruction TCM, configurable from 0KB to 1024KB.
  - Optional instruction cache and data cache, with configurable sizes of 0KB, 4KB, 8KB, 16KB, 32KB, or 64KB.
  - Optional ECC support on the RAM arrays.
- A *Snoop Control Unit* (SCU) that connects each core to the memory system through the AXI3 interfaces.
- An integrated *Generic Interrupt Controller* (GIC) supporting a configurable number of external IRQs, from 0-480 *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, for fault detection.
- High-speed *Advanced Microprocessor Bus Architecture* (AMBA) *Advanced eXtensible Interfaces* (AXI3):
  - A single 64-bit master interface.
  - An optional 64-bit slave *Accelerator Coherency Port* (ACP).
  - An optional second 64-bit master interface, memory mapped and with address filtering support.
  - An optional memory-mapped 32-bit fast peripheral port (FPP) per core, with address filtering.
  - A low-latency memory-mapped 32-bit peripheral port.
  - An optional 64-bit TCM slave port.
  - Optional ECC protection on all AXI interface data and parity on control bits.
- Optional ETM/ATB interface with full instruction and data trace, with either:
  - One ETM, statically shared between each core.
  - Up to four ETMs, one dedicated to each core depending on how many cores are implemented.

1.4 Processor interfaces

Reference information for each of the Cortex-R8 processor external access interfaces.

1.4.1 AXI interfaces

Several AMBA 3.0 AXI interfaces are implemented.

1.4.2 APB Debug interface

AMBA APBv3 is used for debugging purposes.

1.4.3 ETM/ATB interface

You can implement the Cortex-R8 processor with an optional ETM per core, or one optional ETM shared by each core. If an ETM is present, there is a dedicated ETM and *AMBA Trace Bus* (ATB) interface. The ETM provides instruction and data trace for the processor.

Related reference

A.15 ETM signals on page Appx-A-401

Chapter 11 Embedded Trace Macrocell on page 11-238

1.4.4 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 core through a *Cross Trigger Matrix* (CTM).

1.4.5 PMU interface

The Cortex-R8 processor PMU provides eight counters to gather statistics on the operation of the core and memory system. Each counter can count any of the 64 events available in each core.

Related reference

10.1 Performance Monitoring Unit on page 10-215

1.4.6 Test interface

The test interface provides support for test during manufacture of the Cortex-R8 processor using *Memory Built-In Self Test* (MBIST).

See the *Arm® Cortex®-R8 MPCore Processor Integration Manual* for information about the timings of the MBIST signals.

Related reference

A.13 MBIST interface on page Appx-A-396

1.5 Configurable options

List of features of the Cortex-R8 processor that can be configured using either build or pin configurations. 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 cores	One to four	-	Build
Single core with optional redundancy	Single core, no redundancy	One processing core	Build
	Single core with redundancy	Two cores, one built for lock-step mode	Build
Dual core with optional redundancy	Dual core, no redundancy	Two processing cores	Build
	Dual core with redundancy	Two cores, one built for lock-step mode	Build and pin
		Two processing cores	Build and pin
Three cores	Three cores, no redundancy	Three processing cores	Build
Four cores	Four cores, no redundancy	Four processing cores	Build
Instruction cache	No instruction cache[a]	-	Build
	Instruction cache included	No ECC[b] 64-bit ECC	Build
		4KB, 8KB, 16KB, 32KB, or 64KB	Build
Data cache	No data cache[c]	-	Build
	Data cache included	No ECC[b] 32-bit ECC	Build
		4KB, 8KB, 16KB, 32KB, or 64KB	Build
Instruction TCM	No Instruction TCM	-	Build
	Instruction TCM included	No ECC[b] 64-bit ECC	Build
		4KB, 8KB, 16KB, 32KB, 64KB, 128KB, 256KB, 512KB, or 1024KB	Build
Data TCM	No Data TCM	-	Build
	Data TCM included	No ECC[b] 32-bit ECC	Build
		4KB, 8KB, 16KB, 32KB, 64KB, 128KB, 256KB, 512KB, or 1024KB	Build
Branch Target Address Cache (BTAC) size	512 entries	-[b]	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.

Table 1-1 Configurable options (continued)

Feature	Range of options	Sub-options	Build or pin configuration
PREDictor (PRED) RAM size	4096 entries	-[b]	Build
FPU	Not included	-	Build
	Included	Single-precision implementation	Build
		Double-precision implementation	Build
MPU	Number of regions	12 region option	Build
		16 region option	Build
		20 region option	Build
		24 region option	Build
AXI master ports	One port or two ports	Port 0	Build
		Port 1, with address filtering	Build and pin
AXI low-latency peripheral port	Not included	-	Build
	Included	-	Build and pin
AXI fast peripheral port	Not included	-	Build
	Included	One per core	Build and pin
ETM	Not included	-	Build
	Included	One per core	Build
		One shared with all cores	Build
Memory Reconstruction Port (MRP)	Included or not	-	Build
Support for ECC	Used or not	-	Build
Number of interrupts	0-480 in range of 32	-	Build

Related concepts

1.8 Product documentation and design flow on page 1-22

1.6 Redundant processor core comparison

A Cortex-R8 processor that has either only one or only two cores can be implemented with a second, redundant copy of most of the logic. This is known as a dual-redundant configuration. The redundant logic includes a second core that shares the input pins and the cache and TCM of the primary core, so only one set of cache and TCM is required.

The redundant logic includes the second core logic, but not the ETM logic if an ETM is present. The redundant logic operates in lock-step with the primary core, but does not directly affect the primary core behavior in any way. The primary core 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 primary core, 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 primary or redundant logic because of radiation or circuit failure, the comparison logic detects it and asserts the **COMPFAULT** output signal. Used with the RAM error detection schemes, this can help protect the system from faults.
COMPENABLE can be asserted only after an initialization phase, see the *Arm® Cortex®-R8 MPCore Processor Configuration and Sign-off Guide*. See the *Arm® Cortex®-R8 MPCore Processor 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 the Cortex-R8 processor has only two cores and the Split/lock infrastructure implemented, the two core cluster can operate in either split mode or locked mode.

Split mode
Operates as a multiprocessing configuration, with both cores capable of processing, by maintaining L1 data cache coherency. Each core uses its dedicated cache RAM. Also known as performance mode.

Locked mode
Operates in lock-step mode. The second core works as redundant logic for the primary core logic, and the SCU logic, but not the ETM logic if the ETM is present. The redundant core-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 two core processor is held in reset and must remain stable when out of reset.

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

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

1.7 Test features

The Cortex-R8 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. To improve the potential frequency compared to adding multiplexers to the RAM modules, this interface shares some of the multiplexing present in the processor design.

The TCM RAMs can be read and written directly by the program running on the Cortex-R8 processor. You can also use the dedicated AXI3 TCM slave interface to access the TCMs.

Related concepts

12.3 Accessing RAMs using the AXI3 interface on page 12-351

1.8 Product documentation and design flow

The Cortex-R8 MPCore processor is delivered as synthesizable RTL. Before it can be used in a product, the RTL must go through the implementation, integration, and programming design flow processes. The product documentation describes these design flow processes.

1.8.1 Documentation

The Cortex-R8 documentation set includes a Technical Reference Manual (TRM), Configuration and Signoff Guide (CSG), and Implementation Manual (IM).

Technical Reference Manual

The *Technical Reference Manual* (TRM) describes the functionality and the effects of functional options on the behavior of the Cortex-R8 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-R8 processor, then contact:

- The implementer to determine:
  - The build configuration of the implementation.
  - What integration, if any, was performed before implementing the Cortex-R8 processor.
- The integrator to determine the pin configuration of the Cortex-R8 processor that you are using.

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-R8 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 that are used when implementing the Cortex-R8 processor.

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

1.8.2 Design flow

The Cortex-R8 processor is delivered as synthesizable RTL. Before the RTL can be used in a product, it must go through implementation, integration, and programming processes.

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-R8 processor.

The operation of the final device depends on:

Build configuration
: The implementer chooses the options that affect how the RTL source files are preprocessed. 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

Summary of the differences in functionality between product revisions.

r0p0
First release.

r0p1
Fixes errata from r0p0.

r0p2
Set all RAMs on rising edge.

r0p3
Fixes 2 errata from r0p2.

Chapter 2
Functional Description

This chapter describes the functionality of the Cortex-R8 processor.

It contains the following sections:

2.1 About the functions

The Cortex-R8 processor is a highly configurable processor for use in deeply-embedded systems.

2.1.1 Processor block diagram

Figure showing a Cortex-R8 processor design with two cores.

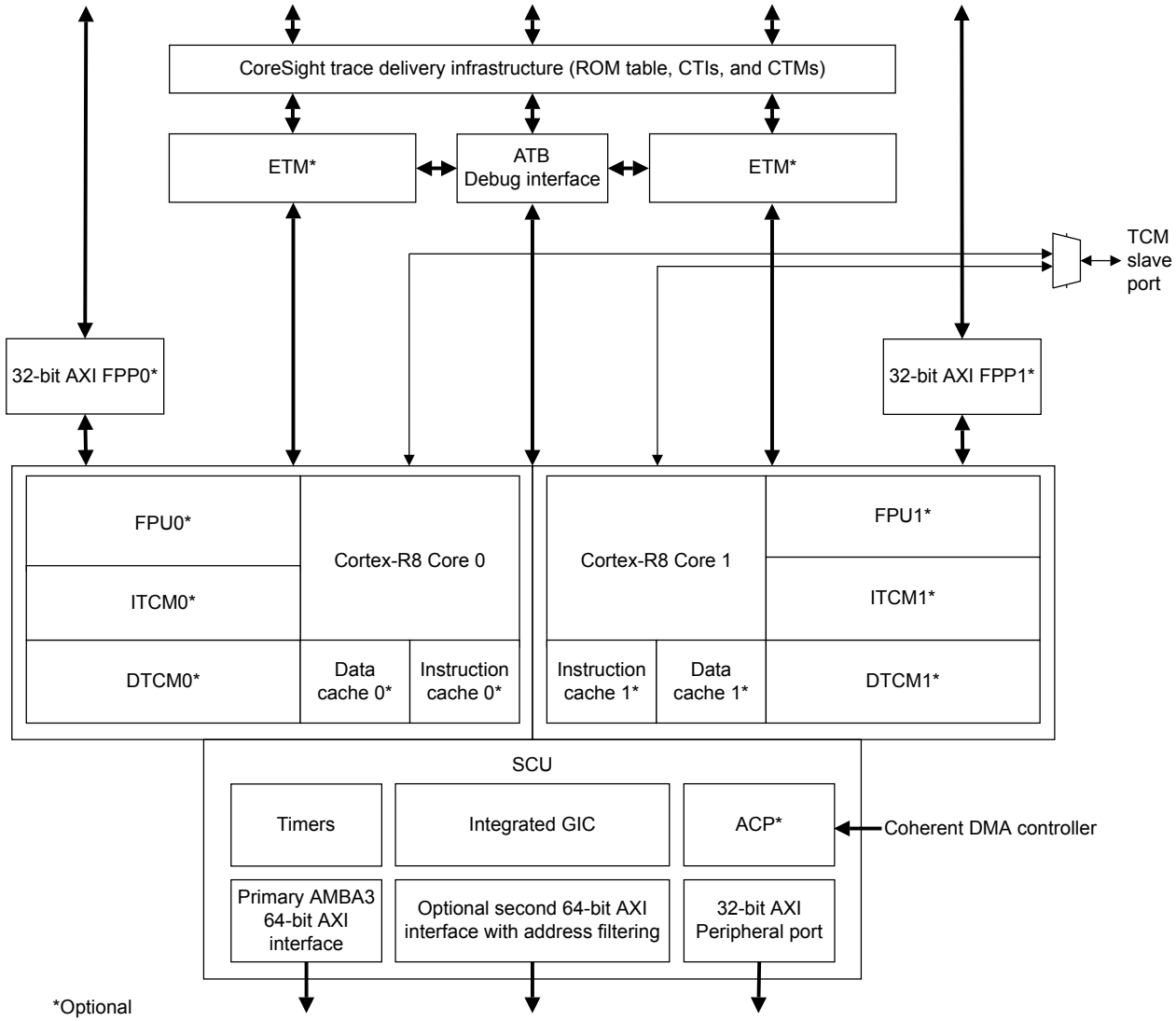


Figure 2-1 Cortex-R8 processor block diagram with two cores

2.1.2 Components of the Cortex®-R8 processor

Functional description of the main components of the Cortex-R8 processor.

L1 memory system

The L1 memory system has 64-bit data paths throughout the memory system, export of memory attributes for external memory systems, and separate optional instruction and data caches each with a fixed line length of 32 bytes.

Snoop Control Unit

The SCU connects the Cortex-R8 processor cores 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).

Note

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

Related concepts

9.1 About multiprocessing and the SCU on page 9-164

2.5.6 AXI slave Accelerator Coherency Port on page 2-48

Interrupt controller

The interrupt controller provides support for handling multiple interrupt sources.

Related reference

9.4 Interrupt controller on page 9-189

Timers

The Timers provide the ability to schedule events and trigger interrupts.

Related concepts

9.6 Global timer on page 9-207

Related reference

9.5 Private timer and watchdog on page 9-201

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.

The debug and trace logic also includes:

- *Performance Monitor Unit* (PMU) based on the PMUv1 architecture.
- *Embedded Trace Macrocell* (ETM) based on the ETMv4 architecture and dedicated ATB interface per core.
- *Cross Trigger Interface* (CTI) and *Cross Trigger Matrix* (CTM) for multicore debugging.

Related reference

10.1 Performance Monitoring Unit on page 10-215

10.3 Embedded Trace Macrocell on page 10-223

10.4 Debug on page 10-224

Split/lock

The Cortex-R8 processor can be configured so that it can be switched, under reset, between a twin-core performance mode and a dual-redundant lock-step mode. This feature imposes extra constraints on the software usage model. Arm provides an `init` code sequence to guarantee the master core and the redundant core leave reset in exactly the same state.

Related concepts

7.7 Static split/lock on page 7-144

2.2 Interfaces

The Cortex-R8 processor provides external interfaces for AXI3, Debug, PMU APB, ATB, DFT, and MBIST controller.

2.2.1 AXI3 interface

The Cortex-R8 processor implements the AMBA 3.0 AXI.

The AMBA 3.0 AXI includes:

- A primary 64-bit master port interface.
- An optional secondary 64-bit master port interface, depending on the build configuration.
- A low-latency peripheral 32-bit master port interface.
- An optional 32-bit fast peripheral port for each core, depending on the build configuration.
- An optional 64-bit ACP slave port, depending on the build configuration.
- An optional 64-bit TCM slave port, depending on the build configuration.

Related concepts

2.5 Processor ports on page 2-42

2.2.2 Debug and PMU APB interface

The Cortex-R8 processor implements an APB slave interface that enables access to Debug and PMU registers, CoreSight components (local ROM table, CTIs, and CTMs), and ETM.

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

2.2.3 ATB interface

The Cortex-R8 processor implements two dedicated ATB interfaces for each core, that output trace information for debugging. The ATB interfaces are compatible with the CoreSight architecture.

See the *Arm® Embedded Trace Macrocell Architecture Specification ETMv4* for more information.

Related reference

A.15 ETM signals on page Appx-A-401

2.2.4 DFT interface

The Cortex-R8 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-R8 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.

Related reference

A.13 MBIST interface on page Appx-A-396

2.3 Clocking, resets, and initialization

Functional description of clocking, resets, and initialization.

This section contains the following subsections:

2.3.1 Clocking

The Cortex-R8 processor includes the **CLK**, **PERIPHCLK**, and **PERIPHCLKEN** clocks. In addition, the DUALPERIPHCLK clock is present if lock-step or split/lock is implemented.

CLK

This is the main clock of the Cortex-R8 processor. All the cores 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

This signal clocks the interrupt controller, global timer, private timers, and watchdog. **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.

The following figure shows an example of clocking the peripherals using these enable signals at a 3:1 ratio.

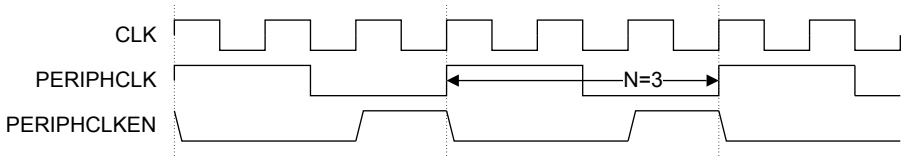


Figure 2-2 Clocking example on MPCore peripherals

DUALPERIPHCLK and DUALPERIPHCLKEN

This clock and clock enable signal are present if lock-step or split/lock is implemented.

Related reference

A.2 Clock and control signals on page Appx-A-358

2.3.2 Resets

The Cortex-R8 processor reset signals enable you to reset parts of your design independently of one another.

In the following list of reset signals, **CN** represents the number of configured cores minus one. For example **DBGACK[CN:0]**, if the number of configured cores is 2, **CN** = **1**.

The Cortex-R8 processor has the following reset signals:

- **CPUCLKOFF[CN:0]**.
- **nCPURESET[CN:0]**.
- **nDBGRESET[CN:0]**.
- **DBGCLKOFF[CN:0]**.
- **nPERIPHRESET**.
- **nSCURESET**.
- **nCPUHALT[CN:0]**.
- **nWDRESET[CN:0]**.
- **WDRESETREQ[CN:0]**.

The following reset signals are present for CoreSight debug logic:

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

The following reset signals are present if an ETM is present. The **x** in the signal name represents **0** for ETM0, **1** for ETM1, **2** for ETM2 or **3** for ETM3:

- **nETMxRESET**.
- **ETMxCLKOFF**.

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

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

The reset signals in the Cortex-R8 processor design enable you to reset different parts of the design independently. The following table shows the supported reset combinations in a Cortex-R8 processor system. [n] refers to the core that receives a reset.

Table 2-1 Reset combinations in a Cortex-R8 processor system

Reset signals	Cortex-R8 processor		Individual cores		Cortex-R8 processor debug	Individual cores	
	Power up	Software	Power up	Software		Debug	Watchdog flag
nSCURESET and **nPERIPHRESET**	0	0	1	1	1	1	1
nCPURESET[CN:0]	All 0	All 0	[n] = 0	[n] = 0	All 1	All 1	All 1
nDBGRESET[CN:0]	All 0	All 1	[n] = 0	All 1	All 0	[n] = 0	All 1
nWDRESET[CN:0]	All 0	All 0	[n] = 0 or all 1	[n] = 0 or all 1	All 1	All 1	[n] = 0

Cortex®-R8 processor power up reset

You must apply power up, or cold, reset to the Cortex-R8 processor when power is first applied to the system. For power up reset, the leading edges, that are the falling edges, of the reset signals do not have to be synchronous to **CLK**, but the rising edges must be. To ensure correct reset behavior, you must assert the reset signals for at least ten **CLK** cycles.

If lock-step or split/lock is enabled, 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**, **nETMxRESET** if an ETM is present.
2. Wait for at least ten **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-R8 processor. You can use an integrated clock gating cell that is driven by a reset controller to stop the **CLK**.
4. Wait for the equivalent of approximately ten cycles, depending on your implementation. This compensates for clock and reset tree latencies.
5. Release all resets.
6. Wait for the equivalent of approximately ten 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**, **nETMxRESET** if an ETM is present.
2. Wait for at least ten **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 ten cycles on every clock domain.
3. Drive **DUALPERIPHCLKOFF**, **SCUCLKOFF**, and **PERIPHCLKOFF** HIGH.
4. Stop the **CLK** clock input to the Cortex-R8 processor.
5. Wait for the equivalent of approximately ten cycles, depending on your implementation. This compensates for clock and reset tree latencies.
6. Release all resets.
7. Wait for the equivalent of approximately ten cycles, to compensate for clock and reset tree latencies.
8. Restart the **CLK** clock and maintain **DUALPERIPHCLKOFF** HIGH.

   SCUCLKOFF and **PERIPHCLKOFF** are driven LOW. You can maintain these two signals HIGH a few more clock cycles, but this is not required.
9. After P* **CLK** cycles, after **PERIPHCLKOFF** is driven LOW, drive **DUALPERIPHCLKOFF** LOW.

——— **Note** ———

P* is defined by the number of delay cycles introduced between the main core and the dual-redundant core. It is configurable and IMPLEMENTATION DEFINED. The default value that is used in the Cortex-R8 processor is two **CLK** cycles.

———————————

Individual core power up reset

This reset applies to an individual core. It initializes the whole logic in a single core, including its debug logic. It is expected to be applied when this core exits from powerdown or dormant state. This reset only applies to configurations where each core is implemented in its own power domain. The **x** at the end of the signal name represents either the core or ETM ID number.

The reset sequence is as follows:

1. Apply **nCPURESETx** and **nDBGRESETx**. If you want to reset the corresponding watchdog flag, you can apply the **nWDRESETx** reset.
2. Wait for at least ten **CLK** cycles, or more if required by other components. There is no harm in applying more clock cycles than this.
3. Assert **CPUCLKOFFx** and **DBGCLKOFFx** with a value of `0b1`.
4. Wait for the equivalent of approximately ten cycles, depending on your implementation. This compensates for clock and reset tree latencies.
5. Release all resets.
6. Wait for the equivalent of approximately ten cycles, to compensate for clock and reset tree latencies.
7. Deassert **CPUCLKOFFx** and **DBGCLKOFFx**. This ensures that all registers in the core see the same **CLK** edge on exit from the reset sequence.

A core power up reset can be extended to enable its corresponding ETM to be powered down. To wake up from powerdown or dormant mode with the ETM, use the **nETMxRESET** and **ETMxCLKOFF** signals in the same way as **nCPURESETx** and **CPUCLKOFFx**.

Note

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

Individual core software reset

This reset applies to an individual core, without debug logic. It initializes all functional logic in a single core apart from its debug logic. All breakpoints and watchpoints are retained during this individual Warm reset. This reset only applies to configurations where each core is implemented in its own power domain. The **x** in the signal name represents either the core or ETM ID number.

Arm recommends that you use the reset sequence for the individual core power up reset. To ensure that the debug registers of the individual cores retain their values, the following signals must not be asserted during the reset sequence:

- **nDBGRESET**
- **nCTRESET**
- **nETMxRESET**

Related concepts

Individual core power up reset on page 2-31

Cortex®-R8 processor debug reset

This reset initializes the debug logic in all cores present in the cluster. To perform a Cortex-R8 processor debug reset, assert all **nDBGRESET** signals for a several **CLK** cycles. **CPUCLKOFFx**, and **ETMxCLKOFF** if an ETM is present, must remain deasserted during this reset sequence.

Individual core debug reset

This reset initializes the debug logic in an individual core in the cluster. To perform an individual core debug reset, assert the corresponding **nDBGRESETx** signal for several **CLK** cycles. **CPUCLKOFFx**, and **ETMxCLKOFF** 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 all cores on the same clock cycle.

Individual core watchdog flag reset

This reset clears the watchdog flag associated with a specific core. Watchdog functionality is independent of all other core functionality, so this reset is independent of 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 all cores in the Cortex-R8 processor. This is because after reset the sticky flag is set in one core, but not the other and this leads to the assertion of **COMPFAULT**. Therefore, if one core has its watchdog flag set, the other core must reach the same state, that is, also having its

watchdog flag set. This can be controlled using the **PERIPHCLKOFF** and **DUALPERIPHCLKOFF** signals.

Related reference

A.2 Clock and control signals on page Appx-A-358

A.3 Reset signals on page Appx-A-360

2.3.3 Input synchronization

All external signals must be synchronous to the **CLK** input.

This is not done internally, therefore synchronization must be done before connecting to the Cortex-R8 processor.

2.3.4 Initialization

Most of the architectural registers in the Cortex-R8 processor, such as r0-r14, and s0-s31 and d0-d15 when floating-point is included, have an unknown value after reset. Therefore, you must initialize these registers 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 Arm®v7-A and Arm®v7-R edition*.

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.

MPU

Before you can use the MPU you must program and enable at least one of the regions, and enable the MPU in the SCTLR.

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.

Related reference

4.3.9 System Control Register on page 4-77

FPU

If the Cortex-R8 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.
- Enable the FPU by setting the EN-bit in the FPEXC register.

Related concepts

5.5.3 Floating-Point Exception Register on page 5-108

Related reference

4.3.11 Coprocessor Access Control Register on page 4-81

Caches

If the Cortex-R8 processor has been built with instruction or data caches, they must be invalidated before they are enabled, otherwise UNPREDICTABLE behavior can occur.

An invalidate all operation never reports any ECC errors.

Related concepts

6.4 About the caches on page 6-117

Related reference

4.3.10 Auxiliary Control Register on page 4-80

TCM

The Cortex-R8 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-R8 processor to use the TCMs from reset.

Preloading TCMs

You can write data to the TCMs using either store instructions or the AXI3 TCM 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 Cortex-R8 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 Cortex-R8 processor APB debug port.

Execute code in debug halt state
: The debug host puts the Cortex-R8 processor into debug halt state and then feeds instructions into it through the *Instruction Transfer Register* (DBGITR). The Cortex-R8 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 the optional AXI TCM slave interface.

Preloading TCMs with ECC

The error code bits in the TCM RAM, if configured with an error scheme, are not initialized by the Cortex-R8 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. You can use the **ITCMECCEN** signal to enable the ITCM when leaving reset.

Before ECC checking is enabled, the entire TCM address space must be initialized with valid data and ECC values. This is because the Cortex-R8 processor can direct speculative accesses into the entire TCM address space at any time.

Note

The entire TCM address range must be initialized as the MPU region programming cannot be used to prevent speculative reads into the TCM.

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-R8 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-R8 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.

Related reference

4.3.10 Auxiliary Control Register on page 4-80

4.3.9 System Control Register on page 4-77

Using TCMs from reset

You can pin-configure the Cortex-R8 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 power up reset.

Related concepts

2.3.2 Resets on page 2-29

Related concepts

8.6 Instruction and data TCM on page 8-161

Related reference

4.3.13 DTCM Region Register on page 4-88

4.3.14 ITCM Region Register on page 4-89

Related reference

4.2.8 c15 registers on page 4-63

2.4 Power management

The Cortex-R8 processor provides mechanisms and support to control both dynamic and static power dissipation.

This section contains the following subsections:

2.4.1 Individual core power management

Placeholders for clamps are inserted around each core to simplify implementation of different power domains.

See the *Arm® Cortex®-R8 MPCore Processor 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 core is shut off so that the core can be seen as non-existent in the cluster. The following table shows the power modes and the wake-up mechanisms for each mode:

Table 2-2 Core power modes

Mode	Description	Wake-up mechanism
Run	The entire device is clocked and powered up.	-
Standby	The core clock is stopped. Only logic required for wakeup is still active.	Standard standby mode wakeup events
Standby mode with RAM retention	The core clock is stopped. Only logic required for wakeup is still active. RAMs are in retention.	
Dynamic RAM retention	Some RAM arrays that are not used or are temporarily disabled are put in retention dynamically.	Normal request from core logic
Dormant	The entire device is powered off except RAM arrays that are in retention mode.	External wakeup event to the power controller, that can perform a reset of the core
Shutdown	The entire device 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 with CPU WFI entry flag to signal to the power controller the power domain that it can cut, using the **PWRCTLOx** bus.

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®-R8 MPCore Processor Configuration and Sign-off Guide* for more information on the interface for RAM retention.

Run mode

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

Standby modes

There are two standby modes in Cortex-R8 processor cores, wait for interrupt, and wait for event.

Wait for Interrupt

Wait for Interrupt (WFI) is a feature of the Armv7-R architecture that puts the core in idle mode by disabling most of the clocks in the core while keeping the core powered up. Only the logic required for wake-up is still active.

WFI reduces the power that is drawn due to the static leakage current, leaving a small clock power overhead to enable the core to wake up from WFI mode. A core enters WFI mode by executing the `WFI` instruction.

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

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

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

Exit from WFI mode occurs when the core detects a reset or one of the WFI wake up events as described in the *Arm® Architecture Reference Manual Arm®v7-A and Arm®v7-R edition*. CP15 broadcasting operations also force an exit from WFI mode.

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

——— **Note** ———

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

———————————

STANDBYWFI continues to assert even if the clocks in the core 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 core in idle mode by disabling most of the clocks in the core while keeping the core powered up. This reduces the power drawn to the static leakage current, leaving a small clock power overhead to enable the core to wake up from WFE mode.

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

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

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

Exit from WFE mode occurs when the core 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 Arm®v7-A and Arm®v7-R edition*. CP15 broadcasting operations also force an exit from WFE mode.

On entry into WFE mode, **STANDBYWFE** for that core is asserted. Assertion of **STANDBYWFE** guarantees that the core is in idle mode. **STANDBYWFE** continues to assert even if the clocks in the core 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 core 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 all cores are in WFI.

A WFI instruction must be executed. The **STANDBYWFI** primary output indicates the WFI mode. A temporary wakeup 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 a core is in WFI, you can exclude the core from the coherency domain. To do this, the state of the core must be saved in the same way as when entering Dormant mode. The core 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®-R8 MPCore Processor Configuration and Sign-off Guide*.

Dynamic RAM retention

Some RAM arrays that are not used or are temporarily disabled, are put in retention dynamically.

The RAM can be:

- The entire instruction cache or data cache.
- The entire ITCM or DTCM.
- A subset of the TCMs, using several 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®-R8 MPCore Processor Configuration and Sign-off Guide*.

Dormant mode

Dormant mode is designed to enable a core 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 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 core is powered down. All RAM blocks must remain powered up during Dormant mode.

Before entering Dormant mode, the state of the core, 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 states must be saved.
- The core must correctly set the CPU Power Status Register in the SCU so that it enters Dormant Mode.
- A Data Synchronization Barrier instruction is executed to ensure that all state saving has been completed.
- The core 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.

Note

If an ETM is present, the **ETMACTIVEx** primary output must be taken into account to ensure that the ETM has completed tracing before powering off the core 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 core until the power is restored. After power is restored, the core leaves reset and, by interrogating the power control register in the SCU, can determine that the saved state must be restored.

The following figure shows the power down sequence.

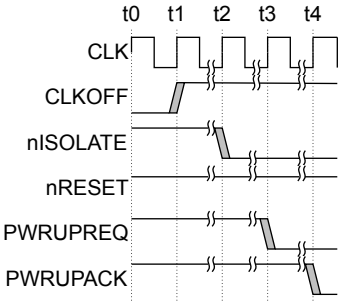


Figure 2-3 Power down sequence

The following figure shows the power up sequence.

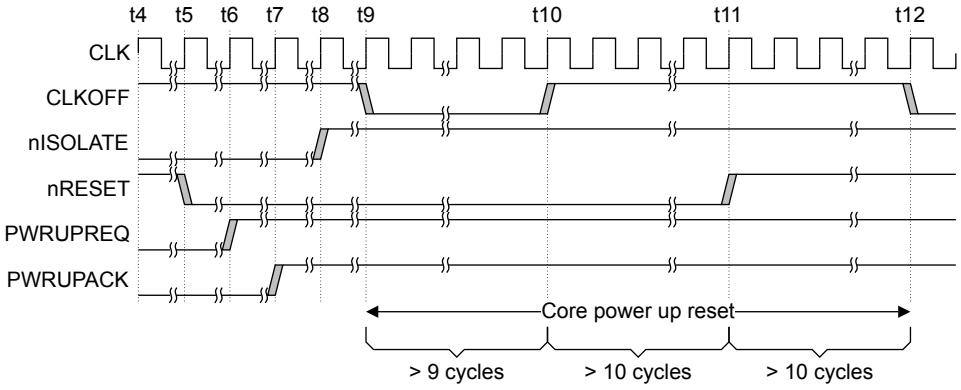


Figure 2-4 Power up sequence

Related reference

9.3.3 SCU CPU Power Status Register on page 9-173

Shutdown mode

Shutdown mode has the entire device powered down, and all states, 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 power down.

Note

If an ETM is present, the **ETMACTIVEx** primary output must be taken into account to ensure that the ETM has completed tracing before powering off the core or the ETM.

Communication to the power management controller

Communication between the core and the external power management controller can be performed using the **PWRCTLOx** Cortex-R8 processor output signals and Cortex-R8 processor input clamp signals.

The **PWRCTLOx** Cortex-R8 processor output signals constrain the external power management controller. The value of **PWRCTLOx** depends on the value of the SCU CPU Power Status Register. The SCU CPU Power Status Register value is only copied to **PWRCTLOx** after the core signals that it is ready to enter low-power mode by executing a WFI instruction and subsequent **STANDBYWFI** output assertion.

Related reference

9.3.3 SCU CPU Power Status Register on page 9-173

Powerdown sequence

To power down a core, perform the following steps:

Procedure

1. Disable the GIC CPU interface for this processor core.
2. Mask IRQ, FIQ, and imp-abort interrupts, with the CPSID I, F, A bits respectively.

3. Execute an `ISB` instruction.
4. Clean and invalidate the data cache.
5. Disable the data cache.
6. Clear the SMP bit in ACTLR. See *4.3.10 Auxiliary Control Register* on page 4-80.
7. Write the power status of the core with value `0x3` in the SCU CPU Power Status Register, with a store-byte.
8. Execute a `DSB` instruction.
9. Execute a `WFI` instruction.

2.4.2 Power domains

The Cortex-R8 processor supports several power domains, including one for each processor, and one for each of the individual processor Cache RAM arrays.

The Cortex-R8 processor can support the following power domains:

- One for each core.
- One for each core Cache RAM array, including the *Branch Predictor* (BP) RAMs.
- One for each core TCM RAM array.
- One for the SCU duplicated tag RAMs.
- One for the remaining logic, the SCU logic cells, and private peripherals.

Note

If an ETM is included for each core, each ETM has its own power domain. In addition, the local CoreSight logic, that is, CTI0 and CTI1, CTI2, CTI3, 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®-R8 MPCore Processor Configuration and Sign-off Guide* for more information.

2.5 Processor ports

Functional description of the Cortex-R8 processor ports.

The Cortex-R8 processor has the following processor ports:

- AXI master port 0.
- AXI master port 1.
- AXI low-latency peripheral port.
- AXI Fast Peripheral Port.
- AXI TCM slave port.
- Accelerator Coherency.
- Memory Reconstruction.
- Private memory region.

This section contains the following subsections:

2.5.1 AXI master port 0

AXI master port 0 has optional ECC protection on data, and parity on control bits. It does not support AXI locked writes, that is, **AWLOCKM0[1]** is always `0b0`.

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 four bits, there are five bits on the AXI master port. If the ACP ID has eight bits, there are nine 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, the core number, or the ACP.

Related concepts

7.4.2 ECC on external AXI bus on page 7-137

2.5.2 AXI master port 1

AXI master port 1 is optional, and has optional ECC protection on data, and parity on control bits. It has an address filtering feature enabled by the SCU Control Register.

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, where the value is defined by the **MFILTERSTART[11:0]** input when leaving reset.
- Master Filtering End Address Register, where the value is defined by the **MFILTEREND[11:0]** input when leaving reset.

The granularity of the mapped memory is 1MB and is defined by the formula:

Memory_space (MB) = End to Start + 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 `0b0`.

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 four bits, there are five bits on the AXI master port. If the ACP ID has eight bits, there are nine 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 low-latency peripheral port. Some L2 cache controllers, such as the CoreLink Level 2 Cache Controller, can enable their own address filtering.

Related concepts

7.4.2 ECC on external AXI bus on page 7-137

Related reference

9.3.1 SCU Control Register on page 9-169

9.3.5 Master Filtering Start Address Register on page 9-176

9.3.6 Master Filtering End Address Register on page 9-177

2.5.3 AXI low-latency peripheral port

The AXI low-latency 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.

The following table shows the AXI low-latency port attributes.

Table 2-3 AXI low-latency port attributes

Attribute	Format
Write issuing capability	15
Read issuing capability	15
Combined issuing capability	30
Write interleave capability	1.

The AXI peripheral port has optional ECC protection on data and parity on control bits.

This port does not support locked writes. **AWLOCKMP[1]** is always `0b0`.

The AXI peripheral port does not support 64-bit accesses, including instruction fetches. 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:

- LLP Filtering Start Address Register, where the value is defined by **PFILTERSTART[11:0]** input when leaving reset.
- LLP Filtering End Address Register, where the value is defined by **PFILTEREND[11:0]** input when leaving reset.

The granularity of the mapped memory is 1MB and is defined by the formula:

Memory_space (MB) = End to Start + 1.

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

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 four bits, there are five bits on the AXI peripheral port. If the ACP ID has eight bits, there are nine 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 overlap, the peripheral port has priority.
- If the address filtering of the fast peripheral port, and the address filtering of the peripheral port overlap, the fast peripheral port has priority.

Related concepts

7.4.2 ECC on external AXI bus on page 7-137

Related reference

9.3 SCU registers on page 9-167

9.3.7 LLP Filtering Start Address Register on page 9-177

9.3.8 LLP Filtering End Address Register on page 9-178

2.5.4 AXI Fast Peripheral Port

Each Cortex-R8 processor core can be implemented with a dedicated fast peripheral port. This is a memory-mapped 32-bit AXI peripheral port that provides fast accesses to a dedicated peripheral.

The following table shows the AXI Fast Peripheral Port attributes.

Table 2-4 AXI Fast Peripheral Port attributes

Attribute	Format
Write issuing capability	8
Read issuing capability	4
Combined issuing capability	12
Write interleave capability	1

The FPP has optional ECC protection on data, and parity on control bits.

The FPP:

- Does not support locked writes, that is, **AWLOCKMFPx[1]** is always `0b0`.
- Does not support 64-bit accesses, including instruction fetches. These accesses always abort. Cacheable accesses, or normal memory accesses through the FPP always abort.
- Does not implement the optional AXI ID fields because all accesses are treated as if they use the same ID.

Address filtering

An FPP access to a core is determined by the FPP filtering start address and FPP filtering end address setup for that core and issued by the FPP dedicated to that core. Accesses that are outside of the address filtering range are directed either to the AXI master ports or to the AXI low-latency peripheral port.

The FPP, LLPP, and master port 1 each have a designated memory-mapped address filtering region. If an access overlaps the address filtering region of two ports, the access is routed to the port with the highest priority. The order of priority of each port is:

- FPP, high priority.
- LLPP.
- Master port 1, low priority.

For example, if the access overlaps the FPP and LLPP regions, the FPP is accessed.

Note

The AXI ordering rules for accesses to the FPP and LLPP are applied independently to each of the two interfaces.

The start and end filtering addresses are configurable in the following SCU registers:

- FPP Filtering Start Address Registers, where the value is defined by the **FPFILTERSTARTx[11:0]** input when leaving reset.
- FPP Filtering End Address Registers, where the value is defined by the **FPFILTERENDx[11:0]** input when leaving reset.

The granularity of the mapped memory is 1MB and is defined by the formula:

Memory_space (MB) = End to Start + 1.

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

Related reference

9.3.14 FPP Filtering Start Address Registers 0-3 on page 9-186

9.3.15 FPP Filtering End Address Registers 0-3 on page 9-187

QoS

When ACTLR.QoS is set for a core, all traffic using the corresponding FPP is treated as high priority. High priority traffic transfers can access either the AXI master port 1 with address filtering enabled, the FPP with address filtering enabled, the AXI low-latency peripheral port, or the data TCM. If the QoS bit is set for several cores, the SCU manages each type of high priority traffic by arbitration. Arm recommends that the QoS bit is set for one core only.

Related concepts

7.4.2 ECC on external AXI bus on page 7-137

2.5.5 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 following table shows the AXI TCM Slave Port attributes.

Table 2-5 AXI TCM Slave Port attributes

Attribute	Format
Write acceptance capability	2
Read acceptance capability	2

The AXI TCM slave port has optional ECC protection on data, and provides parity on control bits.

The AXI TCM slave port also has optional inline single ECC error correction on read accesses for DTCM and ITCM.

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 AXI TCM slave 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 the following table shows.

Table 2-6 AXI TCM slave port access

AxUSERST[2:0]	Access
`0b000`	Instruction TCM for core 0
`0b001`	Data TCM for core 0
`0b010`	Instruction TCM for core 1
`0b011`	Data TCM for core 1
`0b100`	Instruction TCM for core 2
`0b101`	Data TCM for core 2
`0b110`	Instruction TCM for core 3
`0b111`	Data TCM for core 3

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

Limitations of the core and AXI slave port interactions

Do not use the ITCM to store data that is shared with an external master because frequent core read accesses to the ITCM can lock out AXI slave port accesses.

Care must be taken when constructing any mail boxing code where the processor core shares a location in the DTCM with an external master. Any polling of a location by the processor core must be periodic rather than a continuous stream of reads.

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.

Supported AXI transfers

Accesses supported by the AXI TCM slave port.

The following tables show the accesses that the AXI TCM slave port supports.

Table 2-7 Doubleword accesses, aligned on 64-bit address

Signal	Access
AxSIZEST[2:0]	`0x3`
AxLENST[3:0]	Any
AxBURSTST[1:0]	Any (FIXED, INCR, or WRAP)
AxADDRST[2:0]	`0x0`

Table 2-8 Single word accesses, aligned on 32-bit address

Signal	Access
AxSIZEST[2:0]	`0x2`
AxLENST[3:0]	`0x0`
AxBURSTST[1:0]	Ignored
AxADDRST[1:0]	`0x0`

Table 2-9 Single halfword accesses, aligned on 16-bit address

Signal	Access
AxSIZEST[2:0]	`0x1`
AxLENST[3:0]	`0x0`
AxBURSTST[1:0]	Ignored
AxADDRST[0]	`0x0`

Table 2-10 Single byte accesses

Signal	Access
AxSIZEST[2:0]	`0x0`
AxLENST[3:0]	`0x0`
AxBURSTST[1:0]	Ignored

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

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:

- Nonsingle byte, halfword, or word accesses, that is, **AxSIZEST[1:0]** is not `0x3` and **AxLENST[3:0]** is not `0x0`.
- The address for the TCM corresponding to **AxUSERST[1:0]** exceeds the targeted TCM size, defined by **the DTCMRR or ITCMRR** register.
- The access is targeting a TCM of a core that is not present.

The **AxLOCKST**, **AxCACHEST**, and **AxPROTST** signals are not implemented on the AXI TCM slave port. This means that the AXI TCM 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.

Related concepts

7.4.2 ECC on external AXI bus on page 7-137

Related concepts

7.4.2 ECC on external AXI bus on page 7-137

2.5.6 AXI slave Accelerator Coherency Port

The AXI slave Accelerator Coherency Port (ACP) is optional, and has optional ECC protection on data and parity on control bits. The ACP is an AXI3 64-bit slave port that can be connected to noncached AXI3 master peripherals, such as a DMA engine or cryptographic engine.

The following table shows the AXI Accelerator Coherency Port attributes.

Table 2-11 AXI slave Accelerator Coherency Port attributes

Attribute	Format
Write acceptance capability	17 non-shared or 5 shared (without ACP bridge) 18 non-shared or 6 shared (with ACP bridge)
Read acceptance capability	17 non-shared or shared (without ACP bridge) 18 non-shared or shared (with ACP bridge)

This AMBA3 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-R8 processor.

Related concepts

7.4.2 ECC on external AXI bus on page 7-137

9.7 Accelerator Coherency Port on page 9-211

12.6 Accelerator Coherency Port interface on page 12-354

2.5.7 Memory Reconstruction Port

There is an MRP for each core. All write accesses, regardless of their memory attributes, are exported from the core through this port so that an image of the memory can be reconstructed.

Related reference

10.2 Memory Reconstruction Port on page 10-222

2.5.8 Private memory region

All registers accessible by all cores within a Cortex-R8 processor design are grouped into two contiguous 4KB pages accessed through a dedicated internal bus.

The base address of these pages is defined by the configuration signal **PERIPHBASE[31:13]** inputs.

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 core and the Fault Status Register shows this as a SLVERR.

The following table shows the permitted access sizes for the private memory regions.

Table 2-12 Permitted access sizes for private memory regions

Private memory region	Permitted access sizes			
	Byte	Halfword[d]	Word[e]	Doubleword[d]
Global timer, private timers, and watchdogs	No	No	Yes	No
SCU registers	Yes	No	Yes	No
Processor interrupt interfaces				
Interrupt distributor				

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

The following table shows register addresses for the Cortex-R8 processor relative to this base address.

Table 2-13 Cortex-R8 processor private memory region

Offset from PERIPHBASE[31:13]	Peripheral	Description
0x0000-0x00FF	SCU registers	*9.3 SCU registers* on page 9-167
0x0100-0x01FF	Interrupt controller interfaces	*9.4 Interrupt controller* on page 9-189
0x0200-0x02FF	Global timer	*9.6 Global timer* on page 9-207
0x0300-0x03FF	Reserved	Any access to this region causes an SLVERR abort exception
0x0400-0x04FF		
0x0500-0x05FF		
0x0600-0x06FF	Private timers and watchdogs	*9.5 Private timer and watchdog* on page 9-201
0x0700-0x07FF	Reserved	Any access to this region causes an SLVERR abort exception
0x0800-0x08FF		
0x0900-0x09FF		
0x0A00-0x0AFF		
0x0B00-0x0FFF		
0x1000-0x1FFF	Interrupt Distributor	*9.4.4 Distributor register descriptions* on page 9-190

Related concepts

9.6 Global timer on page 9-207

Related reference

A.5 Configuration signals on page Appx-A-362

[d] Halfword or doubleword accesses cause an abort to the requesting core and the Fault Status Register shows this as a SLVERR.
[e] A word access with strobes not all set causes an abort to the requesting core and the Fault Status Register shows this as a SLVERR.

9.3 SCU registers on page 9-167
9.4 Interrupt controller on page 9-189
9.5 Private timer and watchdog on page 9-201
9.4.4 Distributor register descriptions on page 9-190

Chapter 3
Programmers Model

This chapter describes the programmers model.

It contains the following sections:

3.1 About the programmers model

The Cortex-R8 processor implements the Armv7-R architecture.

The Armv7-R architecture 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 Arm®v7-A and Arm®v7-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 the *Arm® Architecture Reference Manual Arm®v7-A and Arm®v7-R edition* for more information about the VFP extension.

Related reference

Chapter 5 Floating Point Unit Programmers Model on page 5-99

3.3 Multiprocessing extensions

The Multiprocessing Extensions are a set of features that enhance multiprocessing functionality.

For implementation-specific details about the Multiprocessing Extensions, see system control and multiprocessing for more information.

See the *Arm® Architecture Reference Manual Arm®v7-A and Arm®v7-R edition* for more information.

Related reference

Chapter 4 System Control on page 4-57
Chapter 9 Multiprocessing on page 9-163

3.4 Memory formats

The Cortex-R8 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-R8 processor can store words in memory as either:

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

See the *Arm® Architecture Reference Manual Arm®v7-A and Arm®v7-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 processor

All addresses are physical addresses. Address translation is not supported.

The instruction and data address spaces are unified.

See the *Arm® Architecture Reference Manual Arm®v7-A and Arm®v7-R edition* for a description of the default memory map, and for more information about memory addresses and access permissions.

Related concepts

6.2 Fault handling on page 6-112

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:

4.1 About system control

The system control coprocessor, CP15, controls and provides status information for the functions implemented in the Cortex-R8 processor. There is one CP15 coprocessor for each core in the Cortex-R8 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.

4.2 Register summary

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.

For more information on using the CP15 system control registers, see the *Arm® Architecture Reference Manual Arm®v7-A and Arm®v7-R edition*.

The following table describes the column headings that the CP15 register summary tables use.

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

This section contains the following subsections:

4.2.1 c0 registers

Summary of 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	0x410FC183	*4.3.1 Main ID Register* on page 4-70
		1	CTR	0x8333C003	Cache Type Register[f]
		2	TCMTR	Implementation dependent[g]	TCM Type Register[f]
		4	MPUIR	Implementation dependent[h]	*4.3.2 MPU Type Register* on page 4-71
		5	MPIDR	Implementation dependent[i]	*4.3.3 Multiprocessor Affinity Register* on page 4-72
		6	REVIDR	Implementation dependent	*4.3.4 Revision ID Register* on page 4-73
	c1	0	ID_PFR0	0x00000131	Processor Feature Register 0[f]
		1	ID_PFR1	0x00000001	Processor Feature Register 1[f]
		2	ID_DFR0	0x00010404	Debug Feature Register[f]
		3	ID_AFR0	0x00000000	Auxiliary Feature Register 0[f]
		4	ID_MMFR0	0x00110130	Memory Model Feature Register 0[f]
		5	ID_MMFR1	0x00000000	Memory Model Feature Register 1[f]
		6	ID_MMFR2	0x01200000	Memory Model Feature Register 2[f]
		7	ID_MMFR3	0x00002111	Memory Model Feature Register 3[f]
	c2	0	ID_ISAR0	0x02101111	Instruction Set Attributes Register 0[f]
		1	ID_ISAR1	0x13112111	Instruction Set Attributes Register 1[f]
		2	ID_ISAR2	0x21232141	Instruction Set Attributes Register 2[f]
		3	ID_ISAR3	0x01112131	Instruction Set Attributes Register 3[f]
		4	ID_ISAR4	0x00010142	Instruction Set Attributes Register 4[f]
1	c0	0	CCSIDR	UNK	*4.3.5 Cache Size ID Register* on page 4-74
		1	CLIDR	Implementation dependent[j]	*4.3.6 Cache Level ID Register* on page 4-75
		7	AIDR	0x00000000	*4.3.7 Auxiliary ID Register* on page 4-76
2	c0	0	CSSELR	Implementation dependent	*4.3.8 Cache Size Selection Register* on page 4-76

4.2.2 c1 registers

Summary of 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	*4.3.9 System Control Register* on page 4-77
		1	ACTLR	0x00000000	*4.3.10 Auxiliary Control Register* on page 4-80
		2	CPACR	0xC0000000	*4.3.11 Coprocessor Access Control Register* on page 4-81

[f] For information, see the *Arm® Architecture Reference Manual Arm®v7-A and Arm®v7-R edition*.
[g] If TCMs are implemented 0x80010001. If TCMs are not implemented 0x00000000.
[h] For 12 MPU regions 0x00000c00. For 16 MPU regions 0x00001000. For 20 MPU regions 0x00001400. For 24 MPU regions 0x00001800.
[i] Dependent on external signal **CLUSTERID** and the number of configured cores in the Cortex-R8 processor.
[j] If cache present 0x09200003. If cache not present 0x00000000.

4.2.3 c5 registers

Summary of 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[k]
		1	IFSR	-	Instruction Fault Status Register[k]

4.2.4 c6 registers

Summary of 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[l]
		2	IFAR	-	Instruction Fault Address Register[l]
	c1	0	DRBAR	UNK	*MPU Region Base Address Registers* on page 4-83
		2	DRSR	`0x00000000`	*MPU Region Size and Enable Registers* on page 4-84
		4	DRACR	UNK	*MPU Region Access Control Registers* on page 4-85
	c2	0	RGNR	UNK	*MPU Memory Region Number Registers* on page 4-87

4.2.5 c7 registers

Summary of the 32-bit wide CP15 system control registers when CRn is c7.

[k] For information, see the *Arm® Architecture Reference Manual Armv7-A and Armv7-R edition*.
[l] For information, see the *Arm® Architecture Reference Manual Armv7-A and Armv7-R edition*.

Table 4-6 c7 register summary

Op1	CRm	Op2	Name	Reset[m]	Description
0	c0	4	NOP	-	No operation[n]
	c1	0	ICIALLUIS	-	Invalidate all instruction caches to PoU Inner Shareable[n],
		6	BPIALLIS	-	Invalidate entire branch predictor array Inner Shareable[n]
	c5	0	ICIALLU	-	Invalidate entire instruction cache[n]
		1	ICIMVAU	-	Invalidate instruction cache by VA to PoU[n]
		4	CP15ISB	-	Instruction Synchronization Barrier operation[n]
		6	BPIALL	-	Invalidate entire branch predictor array[n]
		7	BPIMVA	-	Invalidate MVA from branch predictors[n]
	c6	1	DCIMVAC	-	Invalidate data cache line by VA to PoC[n]
		2	DCISW	-	Invalidate data cache line by Set/Way[n]
	c10	1	DCCMVAC	-	Clean data cache line to PoC by VA[n]
		2	DCCSW	-	Clean data cache line by Set/Way[n]
		4	CP15DSB	-	Data Synchronization Barrier operation[n]
		5	CP15DMB	-	Data Memory Barrier operation[n]
0	c11	1	DCCMVAU	-	Clean data or unified cache line by VA to PoU[n]
	c14	1	DCCIMVAC	-	Clean and invalidate data cache line by VA to PoC[n]
		2	DCCISW	-	Clean and invalidate data cache line by Set/Way[n]

4.2.6 c9 registers

Summary of the 32-bit wide CP15 system control registers when CRn is c9.

[m] These registers do not have a reset value because they are write-only.
[n] For information, see the *Arm® Architecture Reference Manual Arm®v7-A and Arm®v7-R edition*.

Table 4-7 c9 register summary

Op1	CRm	Op2	Name	Reset	Description
0	c1	0	DTCMRR	UNK	*4.3.13 DTCM Region Register* on page 4-88
		1	ITCMRR	UNK	*4.3.14 ITCM Region Register* on page 4-89
	c12	0	PMCR	0x41184000	Performance Monitor Control Register[o]
		1	PMCNTENSET	0x00000000	Count Enable Set Register[o]
		2	PMCNTENCLR	0x00000000	Count Enable Clear Register[o]
		3	PMOVSR	0x00000000	Overflow Flag Status Register[o]
		4	PMSWINC	UNK	Software Increment Register[o]
		5	PMSELR	0x00000000	Event Counter Selection Register[o]
	c13	0	PMCCNTR	UNK	Cycle Count Register[o]
		1	PMXEVTYPER	UNK	Event Selection Register[o]
		2	PMXEVCNTR	UNK	Performance Monitor Count Registers[o]
	c14	0	PMUSERENR	0x00000000	User Enable Register[o]
		1	PMINTENSET	0x00000000	Interrupt Enable Set Register[o]
		2	PMINTENCLR	0x00000000	Interrupt Enable Clear Register[o]

4.2.7 c13 registers

Summary of 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[p]
		2	TPIDRURW	UNK	User Read/Write Software Thread Register[p]
		3	TPIDRURO	UNK	User Read Only Software Thread Register[p]
		4	TPIDRPRW	UNK	Privileged Only Software Thread Register[p]

4.2.8 c15 registers

Summary of the 32-bit wide CP15 system control registers when CRn is c15.

[o] For information, see the *Arm® Architecture Reference Manual Arm®v7-A and Arm®v7-R edition*.
[p] For information, see the *Arm® Architecture Reference Manual Arm®v7-A and Arm®v7-R edition*.

Table 4-9 c15 register summary

Op1	CRm	Op2	Name	Reset	Description
0	c0	0	PCR	0x0	*4.3.15 Power Control Register* on page 4-90
	c1	0	CTDOR	UNK	*4.3.16 Cache and TCM Debug Operation Register* on page 4-91
		1	RADRLO	UNK	*4.3.17 RAM Access Data Registers* on page 4-93, bits[31:0]
		2	RADRHI	UNK	*4.3.17 RAM Access Data Registers* on page 4-93, bits[63:32]
		3	RAECCR[q]	UNK	*4.3.18 RAM Access ECC Register* on page 4-94
	c2	0	D_ECC_ENTRY_0[q]	UNK	*4.3.19 ECC Error Registers* on page 4-95
		1	D_ECC_ENTRY_1[q]	UNK	
		2	D_ECC_ENTRY_2[q]	UNK	
	c3	0	I_ECC_ENTRY_0[q]	UNK	
		1	I_ECC_ENTRY_1[q]	UNK	
		2	I_ECC_ENTRY_2[q]	UNK	
	c4	0	DTCM_ECC_ENTRY[r]	UNK	
	c5	0	ITCM_ECC_ENTRY[r]	UNK	
4	c0	0	CBAR	UNK	*4.3.20 Configuration Base Address Register* on page 4-97

4.2.9 System identification, control, and configuration register

Summary of the system identification, control, and configuration registers.

[q] Only present if ECC is present, otherwise RAZ/WI.
[r] Only present if ECC and TCM are present, otherwise RAZ/WI.

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

Name	CRn	Op1	CRm	Op2	Reset	Description
MIDR	c0	0	c0	0	0x410FC183	*4.3.1 Main ID Register* on page 4-70
CTR				1	0x8333C003	Cache Type Register[s]
TCMTR				2	Implementation dependent[t]	TCM Type Register[s]
MPUIR				4	Implementation dependent[u]	*4.3.2 MPU Type Register* on page 4-71
MPIDR				5	Implementation dependent[v]	*4.3.3 Multiprocessor Affinity Register* on page 4-72
REVIDR				6	Implementation dependent	*4.3.4 Revision ID Register* on page 4-73
ID_PFR0			c1	0	0x00000131	Processor Feature Register 0[s]
ID_PFR1				1	0x00000001	Processor Feature Register 1[s]
ID_DFR0				2	0x00010404	Debug Feature Register[s]
ID_AFR0				3	0x00000000	Auxiliary Feature Register 0[s]
ID_MMFR0				4	0x00110130	Memory Model Feature Register 0[s]
ID_MMFR1				5	0x00000000	Memory Model Feature Register 1[s]
ID_MMFR2				6	0x01200000	Memory Model Feature Register 2[s]
ID_MMFR3				7	0x00002111	Memory Model Feature Register 3[s]
ID_ISAR0			c2	0	0x02101111	Instruction Set Attributes Register 0[s]
ID_ISAR1				1	0x13112111	Instruction Set Attributes Register 1[s]
ID_ISAR2	c0	0	c2	2	0x21232141	Instruction Set Attributes Register 2[s]
ID_ISAR3				3	0x01112131	Instruction Set Attributes Register 3[s]
ID_ISAR4				4	0x00010142	Instruction Set Attributes Register 4[s]
CCSIDR	c0	1	c0	0	UNK[w]	*4.3.5 Cache Size ID Register* on page 4-74
CLIDR				1	Implementation dependent[x]	*4.3.6 Cache Level ID Register* on page 4-75
AIDR				7	0x00000000	*4.3.7 Auxiliary ID Register* on page 4-76
CSSELR		2	c0	0	Implementation dependent	*4.3.8 Cache Size Selection Register* on page 4-76
SCTLR	c1	0	c0	0	-	*4.3.9 System Control Register* on page 4-77
ACTLR				1	0x00000000	*4.3.10 Auxiliary Control Register* on page 4-80
CPACR				2	0xC0000000	*4.3.11 Coprocessor Access Control Register* on page 4-81

4.2.10 Fault handling registers

Summary of the fault handling registers.

s For information, see the *Arm® Architecture Reference Manual Arm®v7-A and Arm®v7-R edition*.
t If TCMs are implemented 0x80010001. If TCMs are not implemented 0x00000000.
u For 12 MPU regions 0x00000c00. For 16 MPU regions 0x00001000. For 20 MPU regions 0x00001400. For 24 MPU regions 0x00001800.
v Dependent on external signal **CLUSTERID** and the number of configured cores in the Cortex-R8 processor.
w Dependent on cache sizes and whether cache is on or off.
x If cache present 0x09200003. If cache not present 0x00000000.

Table 4-11 Fault handling registers

Name	CRn	Op1	CRm	Op2	Reset	Description
DFSR	c5	0	c0	0	-	Data Fault Status Register[y]
IFSR				1	-	Instruction Fault Status Register[y]
DFAR	c6	0	c0	0	-	Data Fault Address Register[y]
IFAR				2	-	Instruction Fault Address Register[y]

4.2.11 MPU registers

Summary of the *Memory Protection Unit* (MPU) registers.

Table 4-12 MPU registers

Name	CRn	Op1	CRm	Op2	Reset	Description
DRBAR	c6	0	c1	0	UNK	*MPU Region Base Address Registers* on page 4-83
DRSR				2	0x00000000	*MPU Region Size and Enable Registers* on page 4-84
DRACR				4	UNK	*MPU Region Access Control Registers* on page 4-85
RGNR			c2	0	UNK	*MPU Memory Region Number Registers* on page 4-87

4.2.12 Cache maintenance operations

Summary of the cache maintenance operations.

[y] For information, see the *Arm® Architecture Reference Manual Arm®v7-A and Arm®v7-R edition*.

Table 4-13 Cache maintenance operations

Name	CRn	Op1	CRm	Op2	Reset	Description
NOP	c7	0	c0	4	-	No operation[z]
ICIALLUIS			c1	0	-	Invalidate all instruction caches to PoU Inner Shareable[z]
BPIALLIS				6	-	Invalidate entire branch predictor array Inner Shareable[z]
ICIALLU			c5	0	-	Invalidate entire instruction cache[z]
ICIMVAU				1	-	Invalidate instruction cache by VA to PoU[z]
CP15ISB				4	-	Instruction Synchronization Barrier operation[z]
BPIALL				6	-	Invalidate entire branch predictor array[z]
BPIMVA				7	-	Invalidate MVA from branch predictors[z]
DCIMVAC			c6	1	-	Invalidate data cache line by VA to PoC[z]
DCISW				2	-	Invalidate data cache line by Set/Way[z]
DCCMVAC			c10	1	-	Clean data cache line to PoC by VA[z]
DCCSW				2	-	Clean data cache line by Set/Way[z]
CP15DSB			c10	4	-	Data Synchronization Barrier operation[z]
CP15DMB				5	-	Data Memory Barrier operation[z]
DCCMVAU			c11	1	-	Clean data or unified cache line by VA to PoU[z]
DCCIMVAC			c14	1	-	Clean and invalidate data cache line by VA to PoC[z]
DCCISW				2	-	Clean and invalidate data cache line by Set/Way[z]

4.2.13 Interface control and configuration registers

Summary of 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	*4.3.13 DTCM Region Register* on page 4-88
ITCMRR				1	UNK	*4.3.14 ITCM Region Register* on page 4-89

4.2.14 Performance monitor registers

Summary of the performance monitor registers.

The following table shows the performance monitor registers.

[z] For information, see the *Arm® Architecture Reference Manual Arm®v7-A and Arm®v7-R edition*.

Table 4-15 Performance monitor registers

Name	CRn	Op1	CRm	Op2	Reset	Description
PMCR	c9	0	c12	0	0x41184000	Performance Monitor Control Register[aa]
PMCNTENSET				1	0x00000000	Count Enable Set Register[aa]
PMCNTENCLR				2	0x00000000	Count Enable Clear Register[aa]
PMOVSR				3	0x00000000	Overflow Flag Status Register[aa]
PMSWINC				4	UNK	Software Increment Register[aa]
PMSELR				5	0x00000000	Event Counter Selection Register[aa]
PMCCNTR			c13	0	UNK	Cycle Count Register[aa]
PMXEVTYPER				1	UNK	Event Selection Register[aa]
PMXEVCNTR			c13	2	UNK	Performance Monitor Count Registers[aa]
PMUSERENR			c14	0	0x00000000	User Enable Register[aa]
PMINTENSET				1	0x00000000	Interrupt Enable Set Register[aa]
PMINTENCLR				2	0x00000000	Interrupt Enable Clear Register[aa]

4.2.15 Miscellaneous system control registers

Summary of 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[ab]
TPIDRURW				2	UNK	User Read/Write Software Thread Register[ab]
TPIDRURO				3	UNK	User Read Only Software Thread Register[ab]
TPIDRPRW				4	UNK	Privileged Only Software Thread Register[ab]

4.2.16 Implementation-defined registers

Summary of the implementation-defined registers.

aa For information, see the *Arm® Architecture Reference Manual Arm®v7-A and Arm®v7-R edition*.
ab For information, see the *Arm® Architecture Reference Manual Arm®v7-A and Arm®v7-R edition*.

Table 4-17 Implementation-defined registers

Name	CRn	Op1	CRm	Op2	Reset	Description
PCR	c15	0	c0	0	0x0	*4.3.15 Power Control Register* on page 4-90
CTDOR			c1	0	UNK	*4.3.16 Cache and TCM Debug Operation Register* on page 4-91
RADRLO				1	UNK	*4.3.17 RAM Access Data Registers* on page 4-93, bits[31:0]
RADRHI				2	UNK	*4.3.17 RAM Access Data Registers* on page 4-93, bits[63:32]
RAECCR[ac]				3	UNK	*4.3.18 RAM Access ECC Register* on page 4-94
D_ECC_ENTRY_0[ac]			c2	0	UNK	*4.3.19 ECC Error Registers* on page 4-95
D_ECC_ENTRY_1[ac]				1	UNK	
D_ECC_ENTRY_2[ac]				2	UNK	
I_ECC_ENTRY_0[ac]			c3	0	UNK	
I_ECC_ENTRY_1[ac]				1	UNK	
I_ECC_ENTRY_2[ac]				2	UNK	
DTCM_ECC_ENTRY[ad]			c4	0	UNK	
ITCM_ECC_ENTRY[ad]			c5	0	UNK	
CBAR		4	c0	0	UNK	*4.3.20 Configuration Base Address Register* on page 4-97

ac Only present if ECC is present, otherwise RAZ/WI.
ad Only present if ECC and TCM are present, otherwise RAZ/WI.

4.3 Register descriptions

Usage constraints, configurations, attributes, and bit assignments for the CP15 system control registers. The registers are described in coprocessor register number order.

This section contains the following subsections:

4.3.1 Main ID Register

The MIDR provides identification information for the Cortex-R8 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 c0 register summary, *4.2.1 c0 registers* on page 4-59.

The following figure shows the MIDR bit assignments.

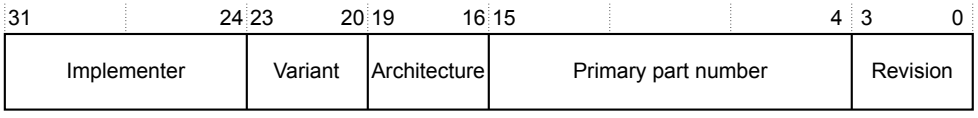


Figure 4-1 MIDR bit assignments

The following table 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 *m* of the r*m*pn 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 *n* of the rmp*n* revision status.

To access the MIDR, read the CP15 register with:

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

Related reference

4.2.1 c0 registers on page 4-59

4.3.2 MPU Type Register

The MPUIR indicates the number of MPU regions, 12, 16, 20, or 24, and 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 c0 register summary, *4.2.1 c0 registers* on page 4-59.

The following figure shows the MPUIR bit assignments.

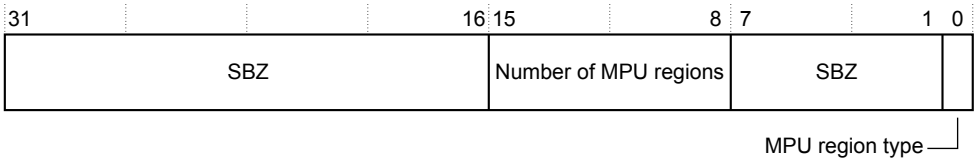


Figure 4-2 MPUIR bit assignments

The following table 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: `0b00011000` 24 regions. `0b00010100` 20 regions. `0b00010000` 16 regions. `0b00001100` 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 `0b0` because the Cortex-R8 processor has unified memory regions. See *3.5 Addresses in the processor* on page 3-56.

To access the MPUIR, read the CP15 register with:

```
MRC p15,0,<Rt>,c0,c0,4 ; Read CP15 MPU Type Register
```

Related concepts

3.5 Addresses in the processor on page 3-56

Related reference

4.2.1 c0 registers on page 4-59

4.3.3 Multiprocessor Affinity Register

The MPIDR 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 c0 register summary, *4.2.1 c0 registers* on page 4-59.

The following figure shows the MPIDR bit assignments.

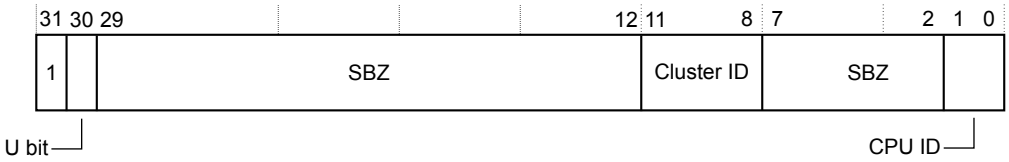


Figure 4-3 MPIDR bit assignments

The following table 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 `0b1`.
[30]	U bit	Multiprocessing Extensions: `0b0` Indicates the Cortex-R8 processor is a multiprocessor configuration, meaning it has several cores.
[29:12]	-	Reserved. SBZ.
[11:8]	Cluster ID	Value read in **CLUSTERID** configuration inputs. It identifies a Cortex-R8 processor in a system that has several Cortex-R8 processors present.
[7:2]	-	Reserved. SBZ.
[1:0]	CPU ID	Indicates the core number in the multiprocessor configuration: `0x00` Core 0. `0x01` Core 1. `0x10` Core 2. `0x11` Core 3.

To access the MPIDR, read the CP15 register with:

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

Related reference

4.2.1 c0 registers on page 4-59

4.3.4 Revision ID Register

The REVIDR 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 c0 register summary, *4.2.1 c0 registers* on page 4-59.

The following figure shows the REVIDR bit assignments.

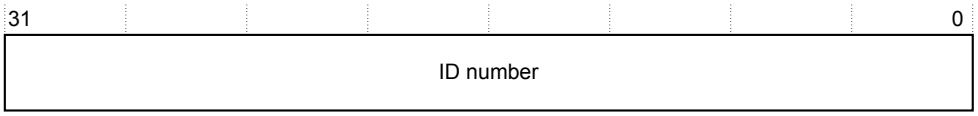


Figure 4-4 REVIDR bit assignments

The following table 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-R8 processor implementation.

To access the REVIDR, read the CP15 register with:

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

Related reference

4.2.1 c0 registers on page 4-59

4.3.5 Cache Size ID Register

The CCSIDR provides information about the architecture of the caches selected by CSSELR.

Usage constraints

The CCSIDR is only accessible in privileged mode.

Configurations

Available in configurations with caches implemented. If caches are not implemented, the value of this register is UNKNOWN.

Attributes

See the c0 register summary, *4.2.1 c0 registers* on page 4-59.

The following figure shows the CCSIDR bit assignments.

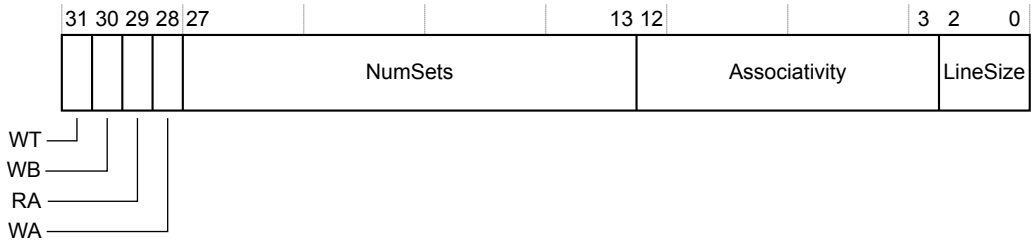


Figure 4-5 CCSIDR bit assignments

The following table shows the CCSIDR bit assignments.

Table 4-22 CCSIDR bit assignments

Bits	Name	Function
[31]	WT	Indicates support available for Write-Through: `0b0` Write-Through support not available.
[30]	WB	Indicates support available for Write-Back: `0b0` Write-Back support not available. `0b1` Write-Back support available.
[29]	RA	Indicates support available for read allocation: `0b0` Read allocation support not available. `0b1` Read allocation support available.
[28]	WA	Indicates support available for write allocation: `0b0` Write allocation support not available. `0b1` 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: **`0b0000000011`** 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`.

Related reference

4.2.1 c0 registers on page 4-59

4.3.8 Cache Size Selection Register on page 4-76

4.3.6 Cache Level ID Register

The CLIDR indicates the cache levels that are implemented in the Cortex-R8 processor and under the control of the System Control Coprocessor. If caches are not implemented, this register value is `0x0`.

Usage constraints

The CLIDR is:

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

Configurations

Available in all configurations.

Attributes

See the c0 register summary, *4.2.1 c0 registers* on page 4-59.

The following figure shows the CLIDR bit assignments.

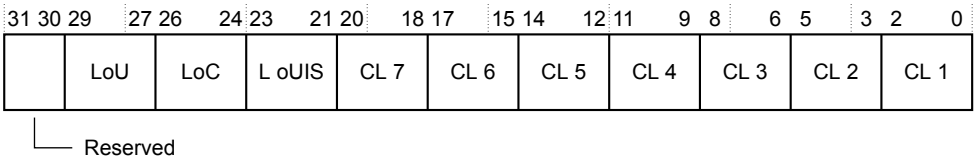


Figure 4-6 CLIDR bit assignments

The following table 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 0b011	Caches not implemented 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
```

Related reference

4.2.1 c0 registers on page 4-59

4.3.7 Auxiliary ID Register

This register is not implemented.

4.3.8 Cache Size Selection Register

The CSSELR selects the current CCSIDR.

Usage constraints

The CSSELR is only accessible in privileged mode.

Configurations

Available in all configurations.

Attributes

See the c0 register summary, *4.2.1 c0 registers* on page 4-59.

The following figure shows the CSSELR bit assignments.

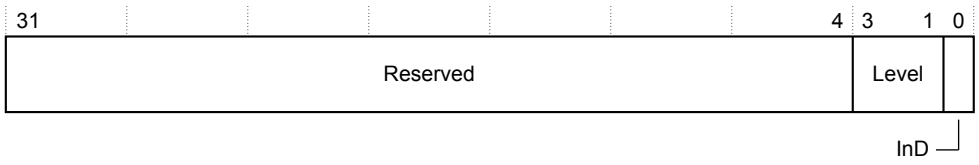


Figure 4-7 CSSELR bit assignments

The following table shows the CSSELR bit assignments.

Table 4-24 CSSELR bit assignments

Bits	Name	Function
[31:4]	-	Reserved. UNP or SBZ.
[3:1]	Level	Cache level selected, RAZ/WI. There is only one level of cache in the Cortex-R8 processor so the value for this field is `0b000`.
[0]	InD	`0b1` Instruction cache. `0b0` 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
```

Related reference

4.2.1 c0 registers on page 4-59

4.3.9 System Control Register

System Control Register (SCTLR) characteristics and bit assignments.

The SCTLR 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.

The following figure shows the SCTLR bit assignments.

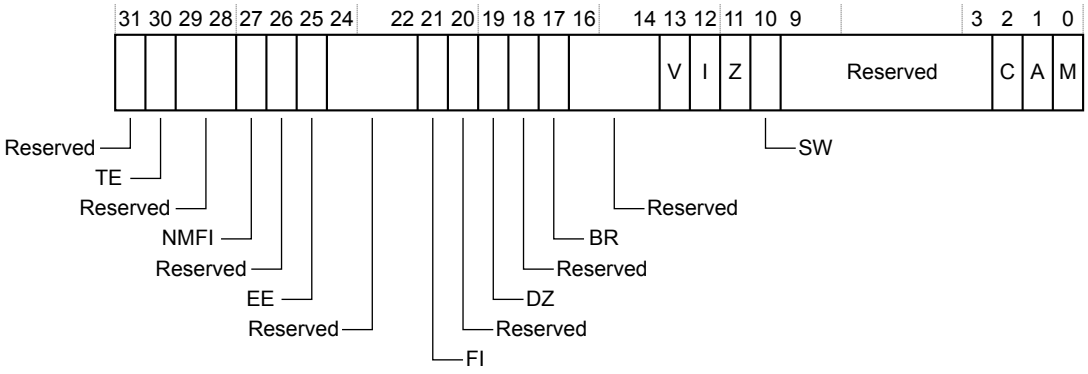


Figure 4-8 SCTLR bit assignments

The following table shows the SCTLR bit assignments.

Table 4-25 SCTLR bit assignments

Bits	Name	Access	Function
[31]	-	-	Reserved. RAZ/SBZP.
[30]	TE	RW	Thumb exception Enable: `0b0` Exceptions including reset are handled in Arm state. `0b1` Exceptions including reset are handled in Thumb state. The **TEINIT** signal defines the reset value.
[29:28]	-	-	Reserved. RAZ/SBZP.
[27]	NMFI	RO	Nonmaskable FIQ support. The bit cannot be configured by software. The **CFGNMFI** signal defines the reset value.
[26]	-	-	Reserved. RAZ/SBZP.
[25]	EE	RW	Exception Endianness. This bit determines how the E bit in the CPSR is set on an exception: `0b0` CPSR E bit is set to `0b0` on an exception. `0b1` CPSR E bit is set to `0b1` on an exception. The **CFGEND** signal defines the reset value.
[24]	-	-	Reserved. RAZ/WI.
[23:22]	-	-	Reserved. 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: `0b0` All performance features enabled. This is the reset value. `0b1` Low interrupt latency configuration. Some performance features disabled.
[20]	-	-	Reserved. 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: `0b0` Divide by zero returns the result zero, and no exception is taken. This is the reset value. `0b1` Attempting a divide by zero causes an UNDEFINED Instruction exception on the `SDIV` or `UDIV` instruction.
[18]	-	-	Reserved. RAO/SBOP.

Table 4-25 SCTLR bit assignments (continued)

Bits	Name	Access	Function
[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: `0b0` Any access to an address that is not mapped to an MPU region generates a Background Fault memory abort. This is the reset value. `0b1` The default memory map is used as a background region: • 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. • An unprivileged access to an address that does not map to an MPU region generates a Background Fault memory abort.
[16]	-	-	Reserved. RAO/SBOP.
[15]	-	-	Reserved. RAZ/SBZP.
[14]	-	-	Reserved. RAZ/WI.
[13]	V	RW	Vectors bit. This bit selects the base address of the exception vectors: `0b0` Normal exception vectors, base address `0x00000000`. `0b1` High exception vectors, Hivecs, base address `0xFFFF0000`. At reset, the value of this bit is taken from **VINITHI**.
[12]	I[ae]	-	Determines if instructions can be cached at any available cache level: **`0b0`** Instruction caching disabled at all levels. This is the reset value. **`0b1`** Instruction caching enabled.
[11]	Z	RW	Enables program flow prediction: `0b0` Program flow prediction disabled. This is the reset value. `0b1` Program flow prediction enabled.
[10]	SW	RW	Swap/Swap Byte (SWP/SWPB) enable bit: `0b0` SWP and SWPB are UNDEFINED. This is the reset value. `0b1` SWP and SWPB perform normally.
[9:7]	-	-	Reserved. RAZ/SBZP.
[6:3]	-	-	Reserved. RAO/SBOP.
[2]	C[ae]	RW	Determines if data can be cached at any available cache level: `0b0` Data caching disabled at all levels. This is the reset value. `0b1` Data caching enabled.

[ae] RW if caches implemented, RAZ/WI if no caches.

Table 4-25 SCTLR bit assignments (continued)

Bits	Name	Access	Function
[1]	A	RW	Enables strict alignment of data to detect alignment faults in data accesses: `0b0` Strict alignment fault checking disabled. This is the reset value. `0b1` 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: `0b0` MPU disabled. This is the reset value. `0b1` MPU enabled.

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

Auxiliary Control Register (ACTLR) characteristics and bit assignments.

The ACTLR controls:
- QoS settings.
- ECC checking, if implemented.
- Allocation in one way.
- Automatic data cache coherency.
- Broadcast of cache, branch predictor, and maintenance operations.
- Enabling the MRP, if implemented.

The following figure shows the ACTLR bit assignments.

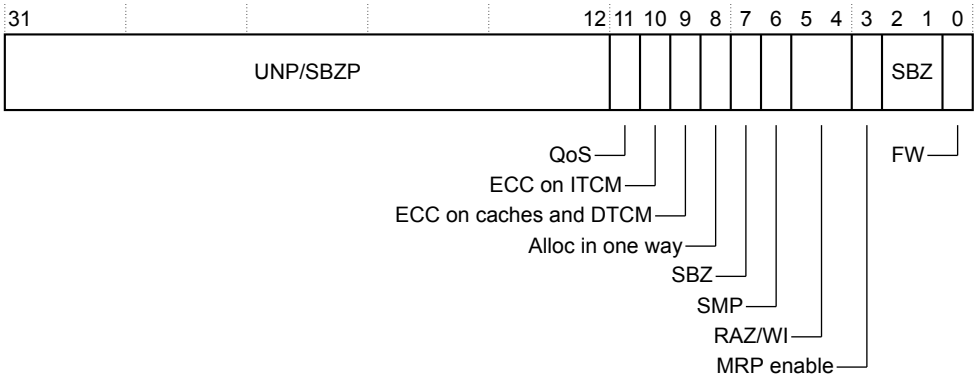


Figure 4-9 ACTLR bit assignments

The following table shows the ACTLR bit assignments.

Table 4-26 ACTLR bit assignments

Bits	Name	Function
[31:12]	-	Reserved. UNP/SBZP.
[11]	QoS	Quality of Service bit: `0b0` Disabled. This is the reset value. `0b1` Enabled. See *8.5 System configurability and QoS* on page 8-159.
[10]	ECC on ITCM	Support for ECC on ITCM, if implemented: `0b0` Disabled. `0b1` Enabled. The reset value is defined by the **ITCMECCEN** 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: `0b0` Disabled. This is the reset value. `0b1` 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]	-	Reserved. SBZ.
[6]	SMP	Signals if the Cortex-R8 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]	-	Reserved. RAZ/WI.
[3]	MRP enable	MRP enable: `0b0` Disabled. This is the reset value. `0b1` Enabled.
[2:1]	-	Reserved. SBZ.
[0]	FW	Cache maintenance broadcast: `0b0` Disabled. This is the reset value. `0b1` Enabled. RAZ/WI if only one core 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

Coprocessor Access Control Register (CPACR) characteristics and bit assignments.

The CPACR:

- 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.

The following figure shows the CPACR bit assignments.

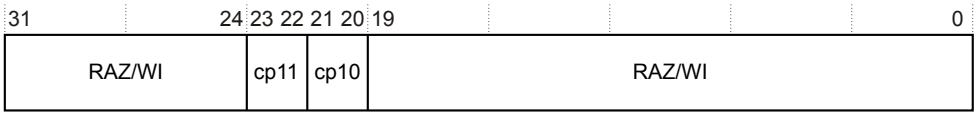


Figure 4-10 CPACR bit assignments

The following table shows the CPACR bit assignments.

Table 4-27 CPACR bit assignments

Bits	Name	Function
[31:24]	-	Reserved. RAZ/WI.
[23:22]	cp11	Defines access permissions for CP11: `0b00` Access denied. This is the reset value, and the behavior for non-existent coprocessors. Attempted access generates an Undefined Instruction exception. `0b01` Privileged mode access only. `0b10` Reserved. `0b11` Privileged and user mode access.
[21:20]	cp10	Defines access permissions for CP10: `0b00` Access denied. This is the reset value, and the behavior for non-existent coprocessors. Attempted access generates an Undefined Instruction exception. `0b01` Privileged mode access only. `0b10` Reserved. `0b11` Privileged and user mode access.
[19:0]	-	Reserved. 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 Arm®v7-A and Arm®v7-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. 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, 16, 20, or 24 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.

MPU Region Base Address Registers

The DRBAR describe the base address of the region specified by the RGNR. The region base address must always align to the region size.

Usage constraints

The DRBAR are only accessible in privileged mode.

Configurations

Available in all configurations.

Attributes

See the c6 register summary, *4.2.4 c6 registers* on page 4-61.

The following figure shows the DRBAR bit assignments.

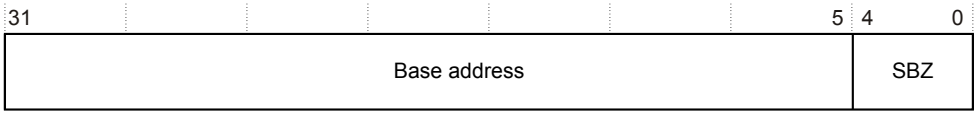


Figure 4-11 DRBAR bit assignments

The following table 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
```

Related reference

4.2.4 c6 registers on page 4-61

MPU Region Size and Enable Registers

MPU Region Size and Enable Registers (DRSR) characteristics and bit assignments.

The DRSR:

- 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.

The following figure shows the DRSR bit assignments.

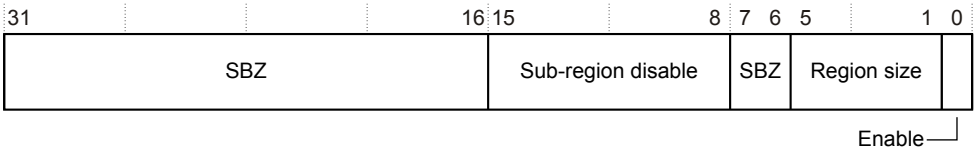


Figure 4-12 DRSR bit assignments

The following table 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[af]. Bit[8] corresponds to sub-region 0. ... Bit[15] corresponds to sub-region 7. The meaning of each bit is: `0b0` Address range is part of this region. `0b1` Address range is not part of this region.
[7:6]	-	Reserved. SBZ.

[af] Sub-region 0 covers the least significant addresses in the region, while sub-region 7 covers the most significant addresses in the region.

Table 4-29 DRSR bit assignments (continued)

Bits	Name	Function		
[5:1]	Region size	Defines the region size: **0b00000-0b00110** UNPREDICTABLE. **0b00111** 256 bytes. **0b01000** 512 bytes. **0b01001** 1KB. **0b01010** 2KB. **0b01011** 4KB. **0b01100** 8KB. **0b01101** 16KB.	0b01110 32KB. 0b01111 64KB. 0b10000 128KB. 0b10001 256KB. 0b10010 512KB. 0b10011 1MB. 0b10100 2MB. 0b10101 4MB. 0b10110 8MB.	0b10111 16MB. 0b11000 32MB. 0b11001 64MB. 0b11010 128MB. 0b11011 256MB. 0b11100 512MB. 0b11101 1GB. 0b11110 2GB. 0b11111 4GB.
[0]	Enable	Enables or disables a memory region: 0b0 Memory region disabled. Memory regions are disabled on reset. 0b1 Memory region enabled. A memory region must be enabled before it is used.		

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.

Related reference

Subregions on page 8-149

MPU Region Access Control Registers

The DRACR registers 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 c6 register summary, *4.2.4 c6 registers* on page 4-61.

The following figure shows the DRACR bit assignments.

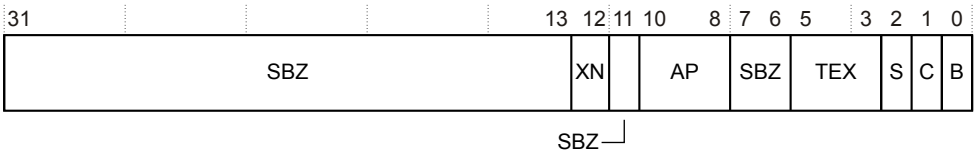


Figure 4-13 DRACR bit assignments

The following table 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: `0b0` All instruction fetches enabled. `0b1` 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 *MPU Region Access Control Registers* on page 4-85.
[7:6]	-	Reserved. SBZ.
[5:3]	TEX	Type extension. Defines the type extension attribute. For more information on this region attribute, see *8.2.3 Region attributes* on page 8-152.
[2]	S	Share. Determines if the memory region is Shareable or Non-Shareable: `0b0` Non-Shareable. `0b1` Shareable. This bit only applies to Normal, not Device or Strongly-Ordered memory.
[1]	C	C bit. For more information on this region attribute, see *8.2.3 Region attributes* on page 8-152.
[0]	B	B bit. For more information on this region attribute, see *8.2.3 Region attributes* on page 8-152.

The following table 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 `0b0`.

Related concepts

8.2.3 Region attributes on page 8-152

Related reference

4.2.4 c6 registers on page 4-61

MPU Region Access Control Registers on page 4-85

MPU Memory Region Number Registers

The RGNR determine which register is accessed. There is one register for each implemented memory region.

Usage constraints

The RGNR are only accessible in privileged mode.

Configurations

Available in all configurations.

Attributes

See the c6 register summary, *4.2.4 c6 registers* on page 4-61.

The following figure shows the RGNR bit assignments.

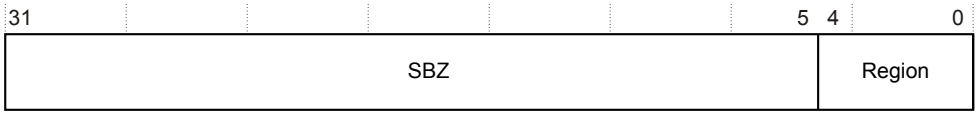


Figure 4-14 RGNR bit assignments

The following table shows the RGNR bit assignments.

Table 4-32 RGNR bit assignments

Bits	Name	Function
[31:5]	-	Reserved. SBZ.
[4:0]	Region	Defines the group of registers to be accessed. Read the MPU Type Register to determine the number of supported regions, see *4.3.2 MPU Type Register* on page 4-71.

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.

Related reference

4.2.4 c6 registers on page 4-61

4.3.2 MPU Type Register on page 4-71

MPU Memory Region Number Registers on page 4-87

4.3.13 DTCM Region Register on page 4-88

4.3.14 ITCM Region Register on page 4-89

4.3.13 DTCM Region Register

The DTCMRR 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 c9 register summary, *4.2.6 c9 registers* on page 4-62.

The following figure shows the DTCMRR bit assignments.

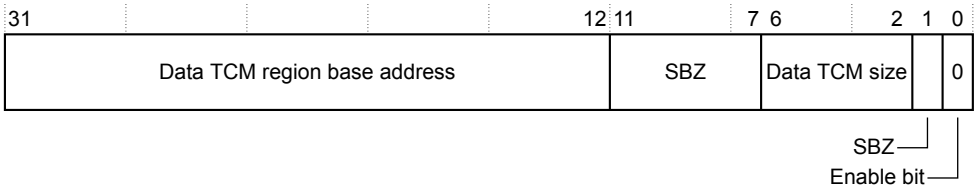


Figure 4-15 DTCMRR bit assignments

The following table 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.
[11:7]	-	Reserved. SBZ.
[6:2]	Data TCM size	Indicates the Data TCM region size: `0b00000` 0KB. `0b00011` 4KB. `0b00100` 8KB. `0b00101` 16KB. `0b00110` 32KB. `0b00111` 64KB. `0b01000` 128KB. `0b01001` 256KB. `0b01010` 512KB. `0b01011` 1024KB. All other values are Reserved.
[1]	-	Reserved. SBZ.
[0]	Enable bit	Enable bit: `0b0` Disabled. This is the reset value. `0b1` Enabled. If Data TCM is not implemented, this field is Read-only and its value is `0b0`.

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
```

Related reference

4.2.6 c9 registers on page 4-62

4.3.14 ITCM Region Register

The ITCMRR Indicates the base address and size of the instruction TCM, and 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 c9 register summary, *4.2.6 c9 registers* on page 4-62.

The following figure shows the ITCMRR bit assignments.

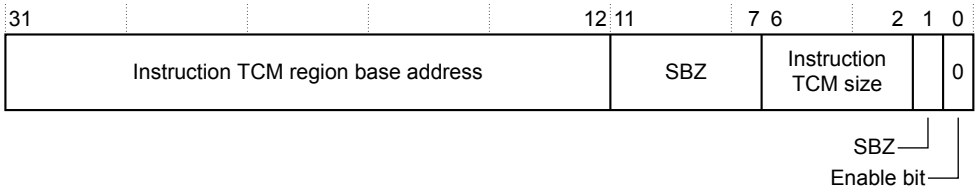


Figure 4-16 ITCMRR bit assignments

The following table 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 **INITRAM0** is HIGH and **VINITH0** is HIGH for core 0, the reset value is `0xFFFF0`, otherwise the reset value is `0x00000`. The same applies for core 1, 2 and 3, if present.
[11:7]	-	Reserved. SBZ.

Table 4-34 ITCMRR bit assignments (continued)

Bits	Name	Function
[6:2]	Instruction TCM size	Indicates the Instruction TCM region size: `0b00000` 0KB. `0b00011` 4KB. `0b00100` 8KB. `0b00101` 16KB. `0b00110` 32KB. `0b00111` 64KB. `0b01000` 128KB. `0b01001` 256KB. `0b01010` 512KB. `0b01011` 1024KB. All other values are Reserved.
[1]	-	Reserved. SBZ.
[0]	Enable bit	Enable bit: `0b0` Disabled. This is the reset value. `0b1` Enabled. When **INITRAM0** is HIGH this enables the Instruction TCM for core 0 directly from reset. The same applies for core 1, 2, and 3 if present. If Instruction TCM is not implemented, this field is Read-only and its value is `0b0`. **Note** If the processor is not configured to include an Instruction TCM, this field must not be set to 1.

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
```

Related reference

4.2.6 c9 registers on page 4-62

4.3.15 Power Control Register

The PCR enables you to set dynamic clock gating.

Usage constraints

There are no usage constraints.

Configurations

Available in all configurations.

Attributes

See the c15 register summary, *4.2.8 c15 registers* on page 4-63.

The following figure shows the PCR bit assignments.

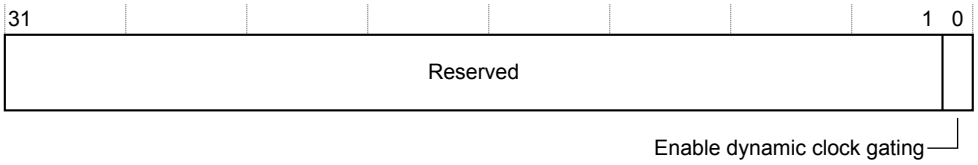


Figure 4-17 PCR bit assignments

The following table 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
```

Related reference

4.2.8 c15 registers on page 4-63

4.3.16 Cache and TCM Debug Operation Register

The CTDOR 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 c15 register summary, *4.2.8 c15 registers* on page 4-63.

The following figure shows CTDOR bit assignments for cache RAMs.

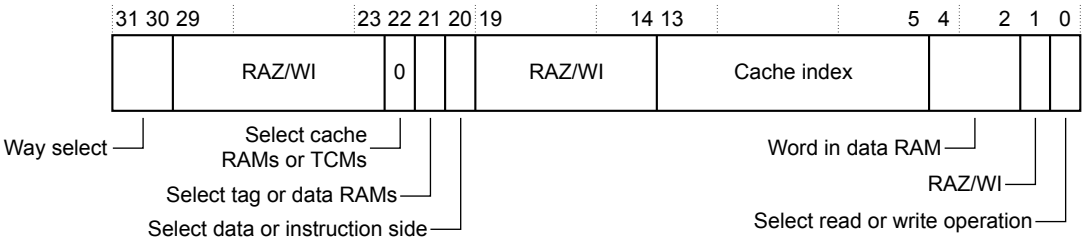


Figure 4-18 CTDOR bit assignments for cache RAMs

The following table shows the CTDOR bit assignments for cache RAMs.

Table 4-36 CTDOR bit assignments for cache RAMs

Bits	Name	Function
[31:30]	Way select	Indicates the way to select in the cache. 0b00 Way 0. 0b01 Way 1. 0b10 Way 2. 0b11 Way 3.
[29:23]	-	Reserved. RAZ/WI.
[22]	Select cache RAMs or TCMs	0b0 Use with cache RAMs.
[21]	Select tag or data RAMs	0b0 Use with tag RAMs. 0b1 Use with data RAMs.
[20]	Select data or instruction side	0b0 Select data side. 0b1 Select instruction side.
[19:14]	-	Reserved. 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. Not required if accessing tag RAMs.
[1]	-	Reserved. RAZ/WI.
[0]	Select read or write operation	0b0 Read operation. 0b1 Write operation.

The following figure shows the CTDOR bit assignments for TCM RAMs.

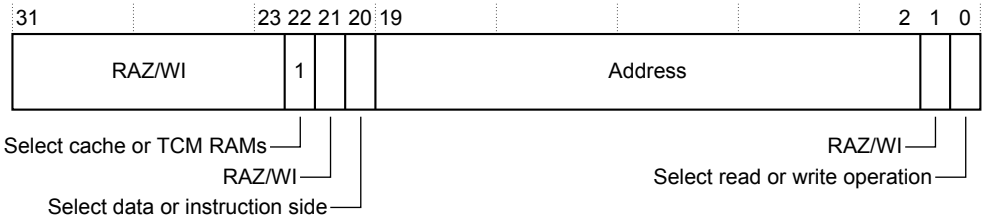


Figure 4-19 CTDOR bit assignments for TCMs

The following table shows the CTDOR bit assignments for TCM RAMs.

Table 4-37 CTDOR bit assignments for TCMs

Bits	Name	Function
[31:23]	-	Reserved. RAZ/WI.
[22]	Select cache or TCM RAMs	0b1 Use with TCMs.
[21]	-	Reserved. RAZ/WI.

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

Bits	Name	Function	
[20]	Select data or instruction side	0b0	Select data side.
		0b1	Select instruction side.
[19:2]	Address	Indicates the address of the TCM RAM.	
[1]	-	Reserved. RAZ/WI.	
[0]	Select read or write operation	0b0	Read operation.
		0b1	Write operation.

To access the CTDOR, write the CP15 register with:

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

Using the CTDOR

Example showing how to use the *Cache and TCM Debug Operation Register* (CTDOR).

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
BIC r1, r1, #1<<20          ; Data side memory selection
ORR r1, r1, #2<<2           ; Word selection
BIC r1, r1, #1<<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
```

Related reference

4.2.8 c15 registers on page 4-63

4.3.17 RAM Access Data Registers

RAM Access Data Registers (RADRLO and RADRHI) characteristics and bit assignments.

The RADRLO and RADRHI:

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 c15 register summary, *4.2.8 c15 registers* on page 4-63.

The following figure shows the RADRLO bit assignments.

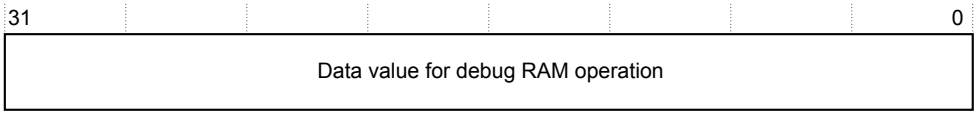


Figure 4-20 RADRLO bit assignments

Note

Bits [31:29] are `0b000` when the tag RAM is read. Only bits [28:0] are meaningful.

The following figure shows the RADRHI bit assignments.

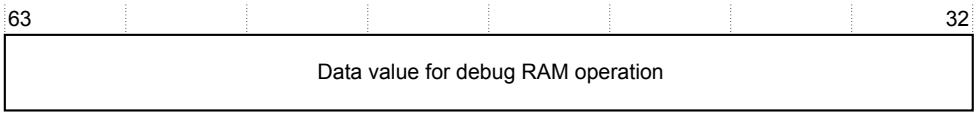


Figure 4-21 RADRHI bit assignments

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
```

Related reference

4.2.8 c15 registers on page 4-63

4.3.18 RAM Access ECC Register

RAM Access ECC Register (RAECCR) characteristics and bit assignments.

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 c15 register summary, *4.2.8 c15 registers* on page 4-63.

The following figure shows the RAECCR bit assignments.

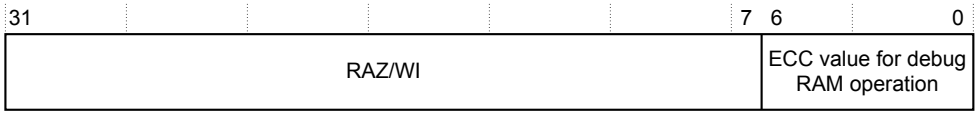


Figure 4-22 RAECCR bit assignments

The following table shows the RAECCR bit assignments.

Table 4-38 RAECCR bit assignments

Bits	Name	Function
[31:7]	-	Reserved. 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
```

Related reference

4.2.8 c15 registers on page 4-63

4.3.19 ECC Error Registers

ECC Error Registers characteristics and bit assignments.

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 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 c15 register summary, *4.2.8 c15 registers* on page 4-63.

The following table 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.

The following table 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.

The following table 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.
[24:20]	Reserved. RAZ/WI.
[19:2]	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 `0b1`. 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 `0b1`, 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.

The following table 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
```

Related reference

4.2.8 c15 registers on page 4-63

4.3.20 Configuration Base Address Register

The CBAR holds the physical base address of the memory-mapped interrupt controller registers.

Usage constraints

The CBAR is a read-only register.

Configurations

Reset to **PERIPHBASE[31:13]** so that software can determine the location of the private memory region.

Attributes

See the c15 register summary, *4.2.8 c15 registers* on page 4-63.

The following figure shows the CBAR bit assignments.

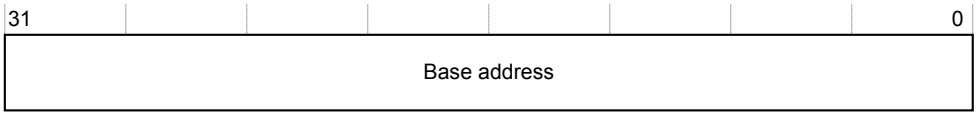


Figure 4-23 CBAR bit assignments

To access the CBAR, read the CP15 register with:

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

Related reference

4.2.8 c15 registers on page 4-63

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:

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 Arm®v7-A and Arm®v7-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 Arm®v7-A and Arm®v7-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

Summary of the issues related to the IEEE 754 standard compliance for hardware and software components, and for software-based components and their availability.

5.2.1 Implementation of the IEEE 754 standard

Implementation choices permitted by the IEEE 754 standard, and list of operations that are not supported.

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 Arm®v7-A and Arm®v7-R edition*.

The following operations are not supported:

- 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 Supported formats

The Cortex-R8 processor supports two build options for the FPU, an optimized FPU is single-precision and half-precision, and 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. However, the timing figures provided here are accurate in most cases. For precise timing, you must use a cycle-accurate model of your processor.

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.

Latency values assume that the instruction has been issued and that the FPU pipeline or the Cortex-R8 processor pipeline are not stalled.

The following table 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	Latency	
			Fwd	Wbck		Fwd	Wbck
FADD FSUB FCVT FSHTOD, FSHTOS FSITOD, FSITOS FTOSHD, FTOSHS FTOSID, FTOSIS FTOSL, FTOUH FTOUI{Z}D, FTOUI{Z}S FTOULD, FTOULS, FUHTOD, FUHTOS FUITOD, FUITOS FULTOD, FULTOS	VADD VSUB VCVT	1	4	4	1	4	4
FMUL FNMUL	VMUL VNMUL	1	4	4	2	6	6

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 FNMAC FMSC FNMSC	VMLA VMLS VNMLS VNMLA	1	7	7	2	9	9
FCPY FABS FNEG FCONST	VMOV VABS VNEG VMOV	1	1	2	1	1	2
FMRS[ag] FMRR(S/D) FMRD(L/H)	VMOV	1 2 1	-	3	1	-	3
FMSR[ah] FM(S/D)RR FMD(L/H)R	VMOV	1	1	2	1	1	2
FMSTAT	VMRS	1	-	3	1	-	3
FDIV	VDIV	10	15	15	20	25	25
FSQRT	VSQRT	13	17	17	28	32	32
FCMP FCMPE FCMPZ FCMPEZ	VCMP VCMP{E} VCMP{E} VCMP{E}	1	1	4	1	1	4
-	FCVT(T/B) .F16.F32	1	2	2	-	-	-
-	FCVT(T/B) .F32.F16	1	-	4	-	-	-

[ag] FPU to Arm.
[ah] Arm to FPU.

5.4 FPU register summary

Summary of 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	`0x41023180`	See *5.5.1 Floating-Point System ID Register* on page 5-106
FPSCR	RW	`0x00000000`	See *5.5.2 Floating-Point Status and Control Register* on page 5-106
MVFR1	RO	`0x01000011`	See the *Arm® Architecture Reference Manual Arm®v7-A and Arm®v7-R edition*
MVFR0	RO	`0x10110221`[ai] `0x10110021`[aj]	See the *Arm® Architecture Reference Manual Arm®v7-A and Arm®v7-R edition*
FPEXC	RW	`0x00000000`	See *5.5.3 Floating-Point Exception Register* on page 5-108

This section contains the following subsections:

- *5.4.1 Processor modes for accessing the FPU system registers* on page 5-104.
- *5.4.2 Accessing the FPU registers* on page 5-105.

5.4.1 Processor modes for accessing the FPU system registers

Summary of 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

[ai] For full FPU implementation, with double precision.
[aj] For single-precision FPU implementation.

5.4.2 Accessing the FPU registers

Access to the FPU registers is controlled by the CPACR.

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

Procedure

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
```

Related reference

4.3.11 Coprocessor Access Control Register on page 4-81

5.5 FPU register descriptions

Characteristics and bit assignments of the FPU system registers.

This section contains the following subsections:

5.5.1 Floating-Point System ID Register

The FPSID Register provides information about the VFP implementation.

Usage constraints

Only accessible in privileged modes.

Configurations

Available in all FPU configurations.

Attributes

See the FPU system registers summary, *5.4 FPU register summary* on page 5-104.

The following figure shows the FPSID Register bit assignments.

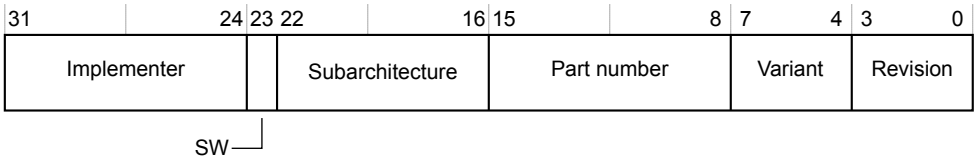


Figure 5-1 FPSID Register bit assignments

The following table 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-R8
[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
```

Related reference

5.4 FPU register summary on page 5-104

5.5.2 Floating-Point Status and Control Register

The FPSCR provides user-level control and status of the FPU.

Usage constraints

There are no usage constraints.

Configurations

Available in all FPU configurations.

Attributes

See the FPU system registers summary, *5.4 FPU register summary* on page 5-104.

The following figure shows the FPSCR bit assignments.

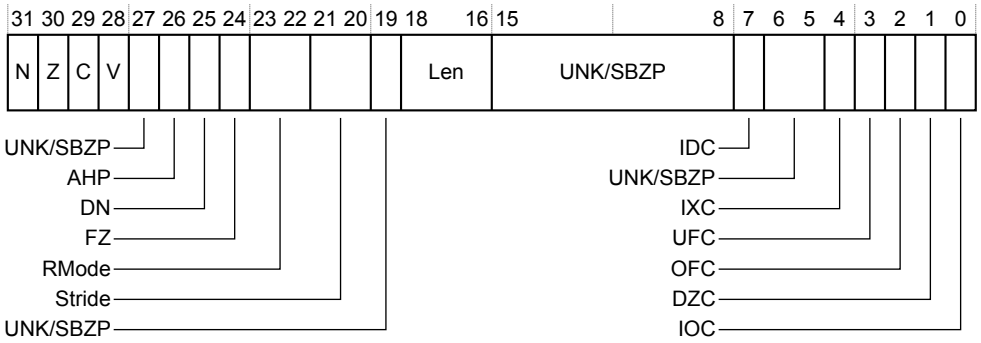


Figure 5-2 FPSCR bit assignments

The following table shows the FPSCR bit assignments.

Table 5-5 FPSCR bit assignments

Bits	Name	Function
[31]	N	Set to `0b1` if a comparison operation produces a less than result.
[30]	Z	Set to `0b1` if a comparison operation produces an equal result.
[29]	C	Set to `0b1` if a comparison operation produces an equal, greater than, or unordered result.
[28]	V	Set to `0b1` if a comparison operation produces an unordered result.
[27]	-	Reserved. 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.

Table 5-5 FPSCR bit assignments (continued)

Bits	Name	Function
[21:20]	Stride	Stride control used for backwards compatibility with short vector values. See the *Arm® Architecture Reference Manual Arm®v7-A and Arm®v7-R edition*.
[19]	-	Reserved. UNK/SBZP.
[18:16]	Len	Vector length, used for backwards compatibility with short vector values. See the *Arm® Architecture Reference Manual Arm®v7-A and Arm®v7-R edition*.
[15:8]	-	Reserved. UNK/SBZP.
[7]	IDC	Input Denormal cumulative exception flag.[ak]
[6:5]	-	Reserved. UNK/SBZP.
[4]	IXC	Inexact cumulative exception flag.[a]
[3]	UFC	Underflow cumulative exception flag.[a]
[2]	OFC	Overflow cumulative exception flag.[a]
[1]	DZC	Division by Zero cumulative exception flag.[a]
[0]	IOC	Invalid Operation cumulative exception flag.[a]

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
```

Related reference

5.4 FPU register summary on page 5-104

5.5.3 Floating-Point Exception Register

The FPEXC Register provides global enable control of the Advanced SIMD and VFP extensions.

Usage constraints

Accessible in all FPU configurations, with restrictions.

Configurations

Available in all FPU configurations.

Attributes

See the FPU system registers summary, *5.4 FPU register summary* on page 5-104.

The following figure shows the FPEXC Register bit assignments.

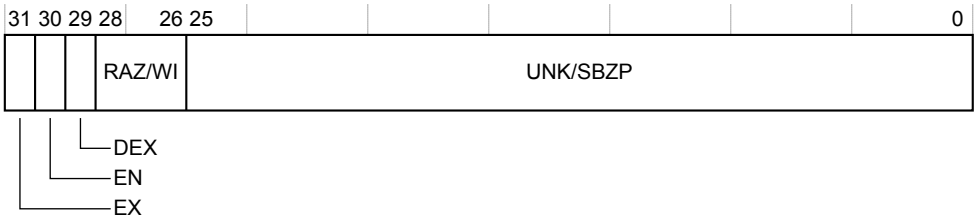


Figure 5-3 FPEXC Register bit assignments

The following table shows the FPEXC Register bit assignments.

[ak] 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 necessary.

Table 5-6 FPEXC Register bit assignments

Bits	Name	Function
[31]	EX	Exception bit: This bit reads-as-zero and ignores writes. The Cortex-R8 processor 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 `0b0` at reset.
[29]	DEX[al]	Defined synchronous instruction exceptional flag: `0b0` No exception has occurred. `0b1` Attempt to perform a VFP vector operation has been trapped[am]. The DEX bit is cleared to `0b0` at reset.
[28:26]	-	Reserved. RAZ/WI.
[25:0]	-	Reserved. UNK/SBZP.

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
```

Related reference

5.4.1 Processor modes for accessing the FPU system registers on page 5-104

5.4 FPU register summary on page 5-104

[al] In single-precision only configurations, this bit is not set for any double-precision operations, regardless of whether they are vector operations or not.

[am] The Cortex-R8 processor FPU hardware does not support the deprecated VFP short vector feature. Attempts to execute VFP data-processing instructions when the FPSCR.LEN field is nonzero 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 necessary.

Chapter 6
Level 1 Memory System

This chapter describes the L1 memory system.

It contains the following sections:

6.1 About the L1 memory system

The processor L1 memory system can be configured during implementation and integration.

L1 memory system consists of:

- Optional separate Harvard instruction and data caches.
- Two optional TCM areas:
  - a configurable *Data TCM* (DTCM) from 0KB to 1024KB with no wait state support.
  - a configurable *Instruction TCM* (ITCM) from 0KB to 1024KB 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.

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.

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 Arm®v7-A and Arm®v7-R edition* for more information.

6.1.1 Data cache policy

Memory regions that are cached in L1 data cache.

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.

Related concepts

8.2 Memory Protection Unit on page 8-148
2.5.7 Memory Reconstruction Port on page 2-48

Related reference

Chapter 7 Fault Detection on page 7-128

6.2 Fault handling

Faults can occur on instruction fetches and on data accesses. When a fault occurs, information about the cause of the fault is recorded in a number of registers, depending on the type of abort. Exactly how you program the Cortex-R8 processor to handle errors depends on the configuration of your processor and system, and what you are trying to achieve.

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.

This section contains the following subsections:

- *6.2.1 Fault classes* on page 6-112.
- *6.2.2 Fault status information* on page 6-113.
- *6.2.3 Usage models* on page 6-114.

6.2.1 Fault classes

The classes of fault that can occur are MPU faults, external faults, debug events, synchronous aborts, and asynchronous aborts.

MPU faults

The MPU can generate an abort for various reasons. 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.

Related concepts

8.2.5 MPU faults on page 8-155

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.

Related reference

10.4 Debug on page 10-224

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-R8 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.

Speculative data fetches that receive an error response from the memory system have the potential to trigger an asynchronous abort. Therefore, to prevent spurious data aborts, the MPU must be programmed to prevent speculative data accesses to memory locations that can return an error response. The memory location should be programmed as one of the following:

- No MPU region defined.
- No access.
- Device memory.
- Strongly ordered memory.

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 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 CSPR 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 several registers, depending on the type of abort.

Abort exceptions

The Link Register, Saved Program Status Register, and Fault Status Register 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 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, *Instruction Fault Status Register* (IFSR) and one for Data Aborts, *Data Fault Status Register* (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 Fault Address Register is updated when a synchronous abort exception is taken.

Fault Address Register

There are two fault address registers, one for Prefetch Aborts, *Instruction Fault Address Register* (IFAR) and one for Data Aborts, *Data Fault Address Register* (DFAR). These indicate the address of the memory access that caused the fault.

6.2.3 Usage models

How you program the Cortex-R8 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.

The following table 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 non-existent 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 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	MPU	Yes	No
Asynchronous external	Any external access	AXI3	No	Yes

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 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 the case of a 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.

Related reference

4.3.19 ECC Error Registers on page 4-95

Debugging cache and TCM access

Refer to the Cache and TCM Debug Operation Register.

Related reference

4.3.16 Cache and TCM Debug Operation Register on page 4-91

Related concepts

8.2.5 MPU faults on page 8-155

6.3 About the TCMs

Instruction and data TCMs are tightly-coupled in the Cortex-R8 processor. There are no external ports for the TCMs and only SRAM memory is supported.

Related concepts

8.6 Instruction and data TCM on page 8-161

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-R8 processor 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.

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.

For more information on the general rules about memory attributes and behavior, see the *Arm® Architecture Reference Manual Arm®v7-A and Arm®v7-Redition*.

This section contains the following subsections:

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.

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

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.

The following table 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

For details about changing these bits, see this TRM for information on disabling or enabling error checking.

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.

Related reference

Disabling or enabling error checking on page 6-124

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 words with an error.

Errors on cache maintenance operations

Details about how the processor handles errors on cache maintenance operations.

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

Organization of the tag RAM and data RAM caches.

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 where:
- The tag RAM bits when ECC is implemented.
- 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.

The following table 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 lookup with one RAM access, all banks selected together. This is done for nonsequential read operations, as shown in the figure for nonsequential read operation performed with one RAM access.
- Select the appropriate bank RAM for sequential read operations, as shown in the figure for sequential read operation performed with one RAM access.
- 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.

The following figure shows a cache lookup being performed on all banks with one RAM access.

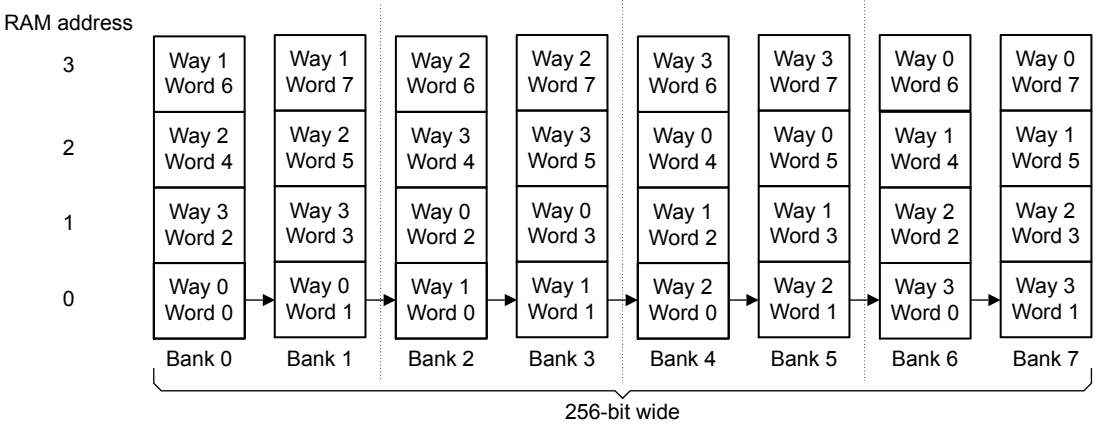


Figure 6-1 Nonsequential read operation performed with one RAM access.

The following figure shows the appropriate bank RAM being selected for a sequential read operation.

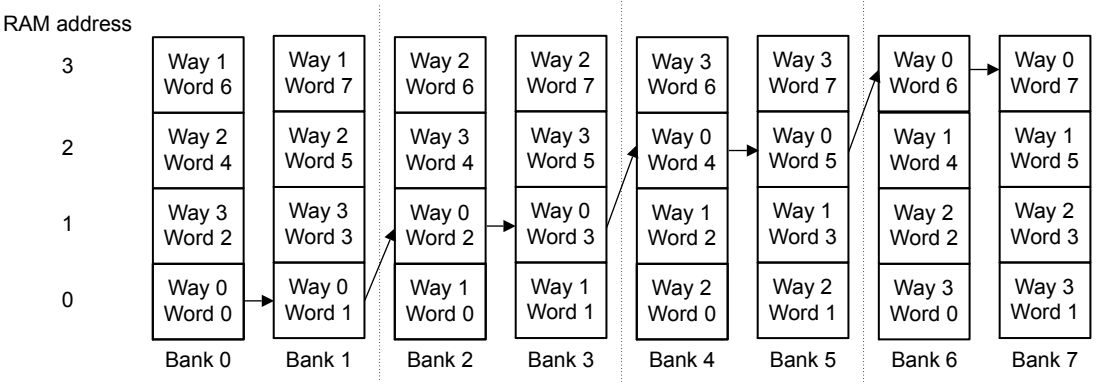


Figure 6-2 Sequential read operation performed with one RAM access

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; that is, without ECC implemented, or with ECC implemented.

Data RAM sizes without ECC implemented

Table showing 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

The following table 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 showing 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

The following table 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

The following table 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

You can enable or disable error checking, and enable or disable the cache RAMs. After you enable or disable the instruction cache, you must issue an `ISB` instruction to flush the pipeline to ensure that all subsequent instruction fetches see the effect of enabling or disabling the instruction cache.

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
```

PLD instruction

The Cortex-R8 processor handles all PLD instructions in a dedicated unit with dedicated resources. This avoids using resources in the integer core or the Load Store Unit.

PLI instruction

The Cortex-R8 processor handles all `PLI` instructions as `NOP`s.

Related concepts

6.4.1 Cache maintenance operations on page 6-117
6.4.2 Cache error detection and correction on page 6-117

6.5 Local exclusive monitor

The processor L1 memory system has a 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 cores. 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 cores. See the *Arm® Architecture Reference Manual Arm®v7-A and Arm®v7-R edition*.

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 core 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-R8 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.

Related reference

Chapter 12 Level 2 Interface on page 12-343

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 multiword 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.
- All Inner Write-Back memory is treated as Write-Back Write-Allocate ignoring any cache allocate hint, though this can dynamically switch to no Write-Allocate, if more than three full cache lines are written in succession.

The following table 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	-[an]	Yes	Yes	Partially[ao]	Yes
	Non-Shareable	Yes	Yes	Yes	Yes	No
Device	Shareable	No	No	No	No[ap]	Yes
	Non-Shareable	No	No	No	No[ap]	Yes
Strongly Ordered	Shareable	No	No	No	No[ap]	Yes

[an] Depends on the value of the ACTLR.SMP bit: 1 = Cacheable. 0 = Non-Cacheable.
[ao] 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.
[ap] Exclusive accesses are handled using both local and global monitor.

6.7 Error detection events

The Cortex-R8 processor implements Arm architectural events.

Related reference

10.1.3 Performance monitoring events on page 10-219

Chapter 7
Fault Detection

This chapter describes the fault detection features of the Cortex-R8 processor.

It contains the following sections:

7.1 About fault detection

The Cortex-R8 processor fault detection features report any detected error to the system, and 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 individual cores is protected by duplication with diagnostic compare. This is known as redundant logic in a lock-step, for a single core configuration, or split/lock, for a two-core configuration, 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 core is responsible for analyzing the notified error and marking the corrupted entries as reusable if it has been proven to be a soft error.

Analysis of errors can be performed as follows:

- Through the MBIST port when setting the cores 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 Cortex-R8 processor uses ECC to indicate errors on the RAMs. It implements error correction using a clean and invalidate and retry for caches, and a correct, writeback, and retry mechanism for TCMs. The processor notifies the detection of any error using primary output events, and the update of performance and statistics counters.

This section contains the following subsections:

7.2.1 Protection method

The methods used by the processor to detect, correct, and report RAM errors.

Detecting errors

The Cortex-R8 processor uses ECC to detect RAM data errors. ECC can also correct a single bit error that might occur in a chunk, where a chunk is typically one or two words of data. However, ECC cannot correct two or more bit errors in the same chunk.

Correcting errors

The Cortex-R8 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, the corresponding index/way is cleaned and invalidated. When the clean and invalidate operation is 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 system containment of multiple-bit ECC errors.

Instruction cache

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 cache and TCMs

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 core causes a clean and invalidate in the corresponding core tag RAM. When the clean and invalidate is done, the line in the SCU tag RAM is marked as unusable.

Related reference

7.2.2 RAM protection summary table on page 7-132

Handling permanent errors

Bank registers are used to mask faulty RAM locations if a hard error occurs. If a processor error occurs, the line is cleaned and invalidated and the ECC error bank prevents any future allocation. For an SCU error, the line is marked as unusable by the SCU error bank but the processor still sees the line as usable.

Permanent errors are handled as follows:

General behavior

If hard, or permanent, errors occur on the RAMs, the clean and 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 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 deadlock. The processor provides a special ECC event indicating the number of corrupted locations 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.

Cortex-R8 is robust to hard-errors, but might require software intervention. When a single-bit error occurs in TCM, the corrected data is written to the error bank and then written back to TCM. The access is then replayed using the error bank data.

If a second single-bit error occurs in the TCM, the error bank is not written to, but corrected data is still written back to the TCM. This allows errors to be isolated. Isolating a hard error prevents its RAM location becoming a double-bit error that is not correctable.

Because subsequent errors do not overwrite the error bank, the replayed access uses the corrected data from the TCM. However, for soft or hard errors:

- If the data written back to TCM has a correctable error, then the data is corrected and used. In this case, no software intervention is required.
- If the data written back to TCM has a hard error, then the data is not corrected and results in permanently corrupted data, causing livelock. In this case, the RAMERR pin goes HIGH continuously.

Note

If you do not apply any software intervention, a soft error can behave as a hard error, and if a second hard error then occurs, it might cause livelock. If a robust to hard errors system is not required, then you do not require software intervention.

Interaction between SCU and cores

For a core 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 core 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 core still sees the line as usable. Therefore, a core 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 core, and stored in the core data error bank.

Related concepts

7.5.3 Error detection notification signals on page 7-140

Reporting errors

The Cortex-R8 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

The table 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: • 25 bits for tag and status. • 9 bits for index.	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: • 23 bits for tag. • 5 bits for index.	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[aq]	64-bit data word	Any error on the data stored in the RAM	None	
BTAC	Parity[ar]	8-bit	-	-	None
PRED	Parity[ar]	1-bit (copy)	-	-	None
SCU tag RAM	SEC-DED ECC	28 bits: • 23 bits for tag. • 5 bits for index.	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

7.2.3 ECC on RAMs

To prevent the loss of any data that might cause the processor or the system to malfunction, the ECC protects the L1 data cache RAMs, L1 instruction cache RAMs, SCU Tag RAM, Instruction TCM, and Data TCM.

RAM targeted

To prevent the loss of any data that might cause the Cortex-R8 processor or the system to malfunction, the ECC protects the L1 data cache RAMs, L1 instruction cache RAMs, SCU tag RAM, Instruction TCM, and Data TCM.

Related reference

7.2.2 RAM protection summary table on page 7-132

aq The SEC-DED ECC is used as a Double Error Correction because the lines are clean.

ar 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.

Basic scheme

The basic ECC scheme for the L1 cache, SCU RAM, and TCM is that any error, either soft or hard, is indicated to the system, and any error detected is stored in an error bank, waiting to be analyzed by the system, and preventing that location from being used.

In addition:

- The system analyzes the faulty RAM location to check the full RAM space (MBIST).
- The core 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 core 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.

The following table shows the basic scheme, and also the differences where it is part of the core, 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[as]	-	Error notification[as]
Uncorrectable error notification to the system	Error notification[as]	Error notification[as]	Error notification[as]
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 core 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

Related concepts

7.1.2 Analysis of errors on page 7-129

Related reference

A.11 Error detection notification signals on page Appx-A-384

Auto-check mechanism

The direct access to the faulty RAM location by the core 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.

Related reference

9.3.12 SCU Debug tag RAM access on page 9-182

[as] Fault detection error notification is done by the primary output pins.

4.3.16 Cache and TCM Debug Operation Register on page 4-91

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 codes

To protect the different types of RAM, 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.

7.2.5 RAM configuration

RAM configuration for ECC and parity.

The following table 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	4 × 29 bits	4 × (29 + 7) bits
Data data RAM	8 × 32 bits	8 × (32 + 7) bits
Instruction tag RAM	4 × 23 bits	4 × (23 + 7) bits
Instruction data RAM	4 × 64 bits	4 × (64 + 8) bits
SCU tag RAMs	4 × 23 bits	4 × (23 + 7) bits
Data TCM	2 × 32 bits	2 × (32 + 7) bits
Instruction TCM	4 × 64 bits	4 × (64 + 8) bits

The following table 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	2 × 32 + 2 × 28 bits	2 × (32 + 4) + 2 × (28 + 4) bits
PRED	4 × 4 bits	4 × (4 + 4) bits

7.2.6 Performance impact

In an error-free system, the major performance impact is the cost of the read-modify-write scheme for nonfull 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

A Cortex-R8 processor that has either one or two cores can have optional logic protection. When the processor is configured with a single core and logic protection, it is implemented as a Lock-Step configuration. When the processor is configured with two cores and logic protection, it is configured as a Split-Lock configuration.

The logic of the primary core is protected by a duplicate core that is the exact copy of the first core. Both cores share the same RAMs protected with ECC and the same input pins. The second core is delayed by two clock cycles so that this redundant system can detect glitches in the inputs.

The outputs of the primary core and the duplicate core are compared on each cycle to detect any error. The outputs of the first core are delayed so they can be synchronized with the second core. This mechanism relies on the fact that any error occurring in the core is eventually visible on the outputs of the core, or is inherently a low-risk failure.

On detection of an error in one core, both cores 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 Cortex-R8 processor provides a template of the logic required for the comparison of the dual-redundancy cores.

Related concepts

7.6 Lock-step on page 7-141
7.7 Static split/lock on page 7-144

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 AXI low-latency peripheral port.
- The AXI fast peripheral port(s).
- The optional AXI ACP slave port.
- The AXI TCM 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, or a parity error on a control bit.

AXIFATALERR
A fatal ECC error has been detected on one of the channels.

It is also assumed that any bus signal always has a defined value after reset.

7.4.2 ECC on external AXI bus

All master buses, that is, AXI master port 0 and AXI master port 1, AXI low-latency peripheral port, AXI fast peripheral port, and the optional AXI TCM slave 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.
- 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.

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 with ACP or TCM, when byte line strobes are sparse on an ACP or a TCM slave port 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 or the TCM slave port must compute the ECC bits together with the write data with the same assumption.

If ECC errors are found on TCM RAMs:

- For reads, the slave error is reported for a fatal error. If the error is fatal, the **FATALRAMERR** output is also raised. If the error is correctable, it is corrected internally and no error is reported.
- 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-R8 processor.

Note

To enable ECC in the Cortex-R8 processor, you must first enable ECC on the RAMs.

Related reference

9.7.6 ACP bridge on page 9-212

7.5 Programmers view

Various processor and SCU registers are used to enable and monitor ECC. When an error is detected, error signals are asserted to notify the external system responsible for analyzing the fault.

This section contains the following subsections:

7.5.1 Processor registers

Processor registers used in ECC.

Auxiliary Control Register (ACTLR)
Bits[10:9] of this register are used to enable ECC checking.

ECC Error Registers (DEER0-2/IEER0-2 and DTCMEER/ITCMEER)
These registers provide information on ECC errors.

Performance counters
The performance counters can be configured to monitor several ECC-related metrics.

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.

Related reference

4.3.10 Auxiliary Control Register on page 4-80

4.3.19 ECC Error Registers on page 4-95

10.1.3 Performance monitoring events on page 10-219

4.3.16 Cache and TCM Debug Operation Register on page 4-91

7.5.2 SCU registers used in ECC

The SCU is seen as a peripheral by the Cortex-R8 processor core(s), 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.

SCU Error Bank Registers
Bits[13:5] of these registers hold the SCU tag RAM index, and bits[1:0] show the error status.

Performance counters
Events related to the SCU are reported to the PMU of each core. The performance counters can be configured to monitor several ECC-related metrics.

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 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.

Related reference

9.3.1 SCU Control Register on page 9-169
9.3.10 SCU Error Bank First Entry Register on page 9-180
9.3.11 SCU Error Bank Second Entry Register on page 9-181
10.1.3 Performance monitoring events on page 10-219
9.3.12 SCU Debug tag RAM access on page 9-182

7.5.3 Error detection notification signals

When an error is detected, error signals are asserted to notify the external system responsible for analyzing the fault.

Related reference

A.11 Error detection notification signals on page Appx-A-384

7.6 Lock-step

Lock-step mode requires the Cortex-R8 processor to be implemented with a second, redundant copy of the `cpu_noram`, `scu_noram`, and `axis` modules. This provides redundancy in the logic without duplicating the RAMs that are protected by ECC.

The following figure shows how lock-step is implemented.

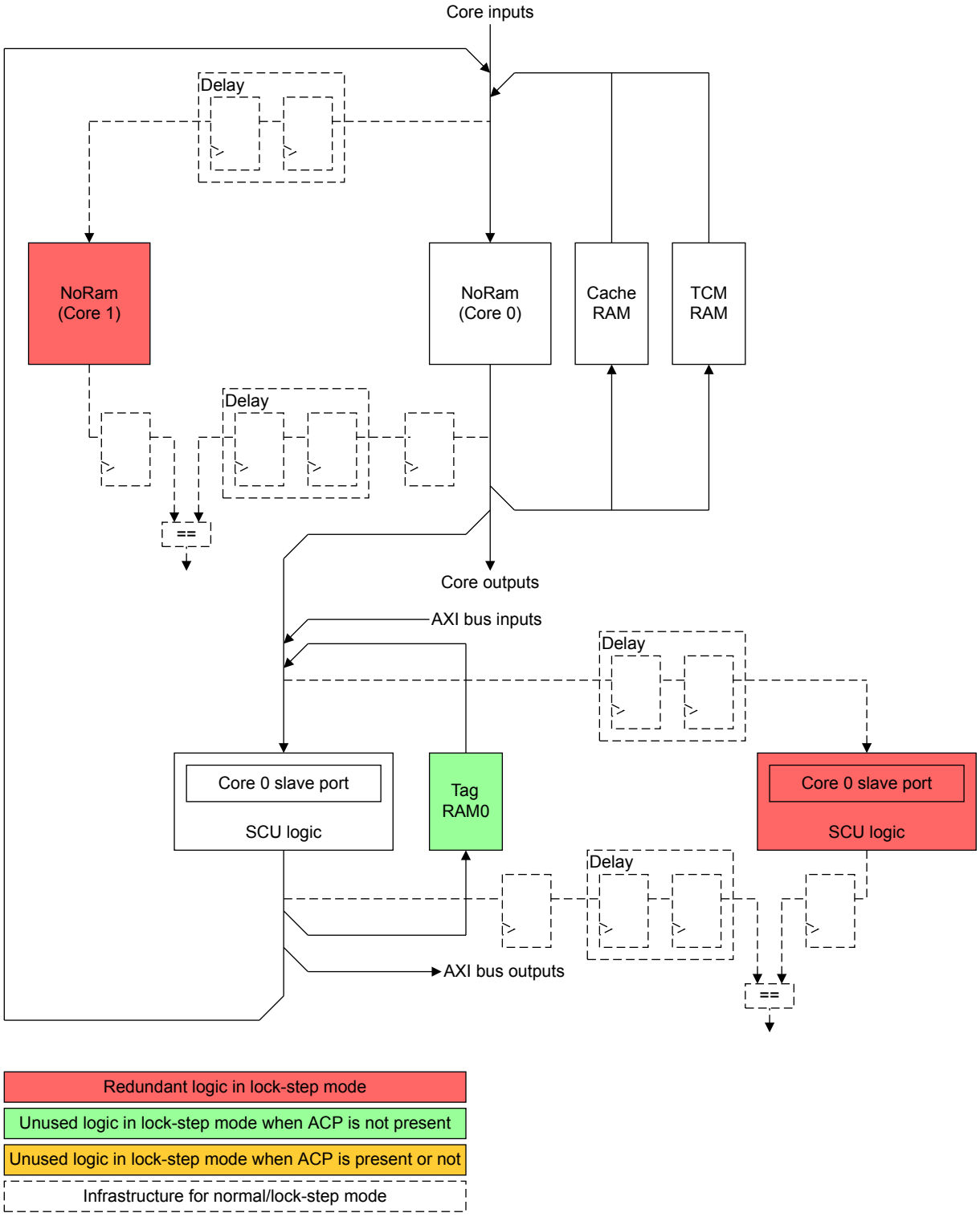


Figure 7-1 Lock-step

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 core 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, and lock-step mode, that is, a lock-step configuration with dualredundant 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.

The following figure shows how split/lock is implemented.

In static split/lock, core 1 is present with both `noram` and `ram` modules. When the Cortex-R8 processor is operating in lock-step mode, the `ram` logic is clamped.

Only the core 0 slave port of the SCU is duplicated. The interface with core 1 is not required because, when in lock-step mode, the core 1 `noram` is used as a redundancy of core 0.

Note

You can implement static split/lock in a Cortex-R8 processor that has only two cores.

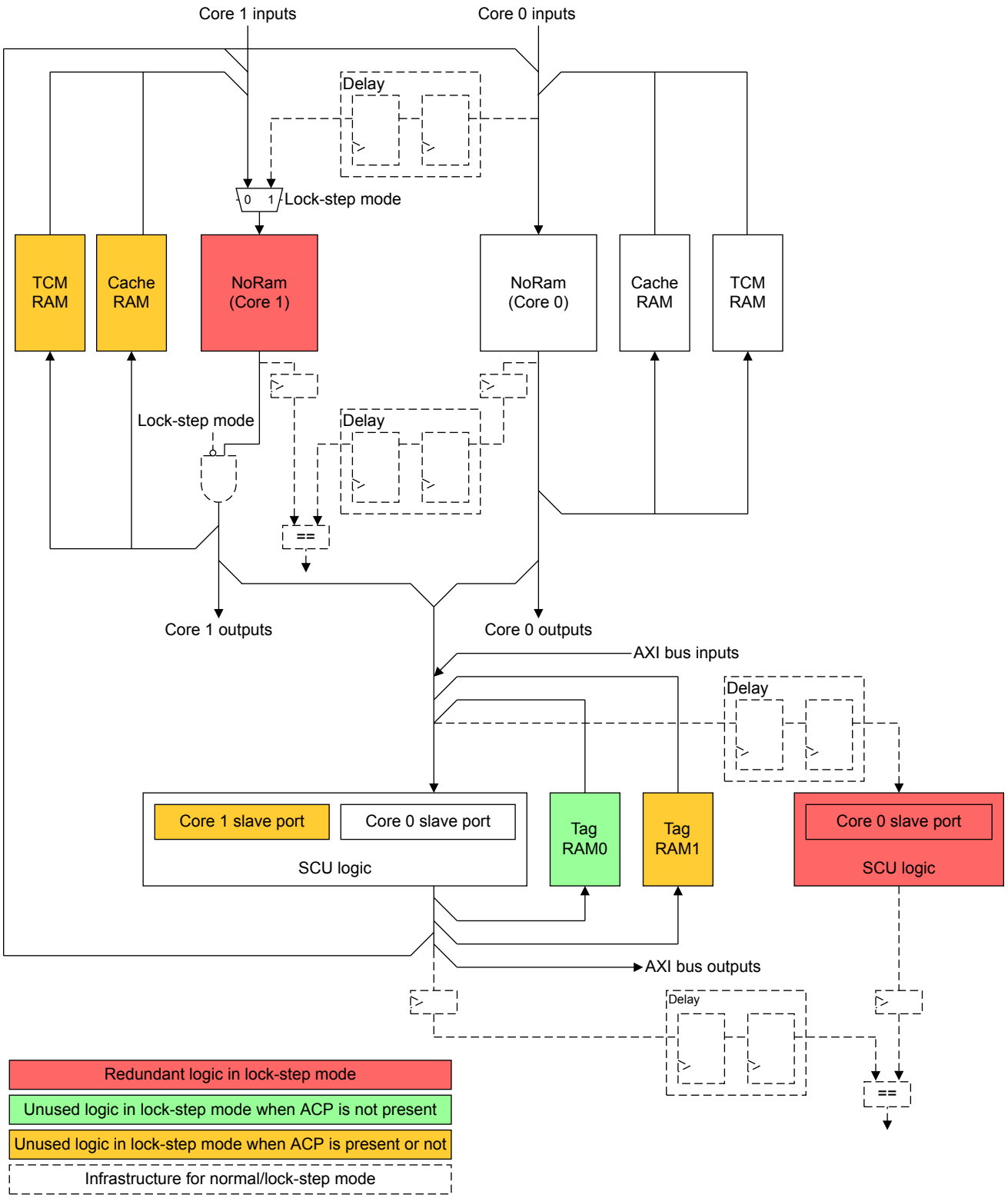


Figure 7-2 Static split/lock

Chapter 8
Determinism Support

This chapter describes the determinism support features of the Cortex-R8 processor.

It contains the following sections:

8.1 About determinism support

The Cortex-R8 processor contains several features that provide deterministic timing and low interrupt latency for hard real-time applications. These features are in each core 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. The MPU enables you to partition memory into regions and set individual protection attributes for each region.

The MPU is programmed using CP15 registers c1 and c6. Memory region control read and write access is permitted only from privileged modes.

The MPU enables you to partition memory into regions and set individual protection attributes for each region. The MPU supports 12, 16, 20, or 24 memory regions, depending on your implementation.

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. For an MPU that has 16 memory regions, region 15 has the highest priority. The MPU returns access permissions and attributes for the highest priority region where the address hits.

For a full architectural description of the MPU and the memory map, see the *Arm® Architecture Reference Manual Arm®v7-A and Arm®v7-R edition*.

This section contains the following subsections:

8.2.1 Regions

The MPU regions are memory regions, overlapping regions, background regions, and TCM regions.

Memory regions

The MPU can have 12, 16, 20, or 24 regions, depending on the implementation. For each memory region you can define the region base address, size, access permissions, and region attributes. Each region can be split into eight equal-sized non-overlapping subregions.

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. See the DRSR bit assignments for the encoding.

Related reference

A.11.2 RAM ECC error bank status signals on page Appx-A-384

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 can have attributes assigned for Memory type, Shareable or Non-Shareable, Non-Cacheable, and Write-Back Write Allocate.

Related concepts

8.2.2 Memory types on page 8-151
8.2.3 Region attributes on page 8-152

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.

For more information, see the *Arm® Architecture Reference Manual Arm®v7-A and Arm®v7-R edition*. For access data permission bit encodings, see the information for the MPU Region Access Control Registers.

Related reference

MPU Region Access Control Registers on page 4-85

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. In an MPU with 12 regions, attributes and permissions for region 11 take highest priority, those for region 0 take lowest priority.

For example:

Region 2
: Is 4KB in size, starting from address `0x3000`. Privileged mode has full access, and user mode has read-only access.

Region 1
: Is 16KB in size, starting from address `0x0000`. Both privileged and user modes have full access.

When the core performs a data write to address `0x3010` while in user mode, the address falls into both region 1 and region 2, as the following figure 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.

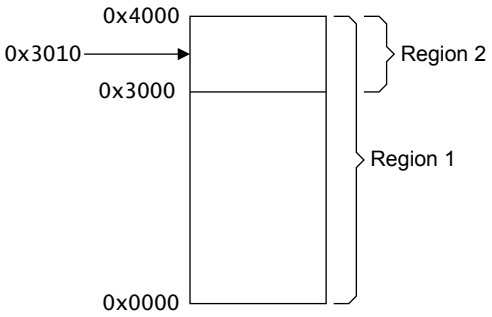


Figure 8-1 Overlapping memory regions

Example of using regions that overlap

You can use overlapping regions for stack protection.

For example: If the current process overflows the stack it uses, a write access to region 2 by the core causes the MPU to raise a permission fault.

- 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.

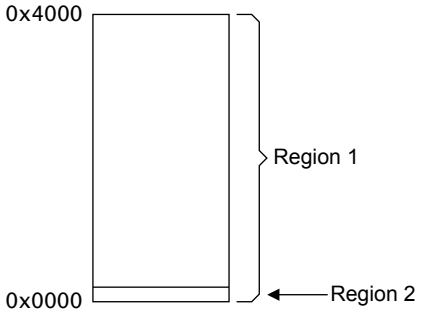


Figure 8-2 Overlay for stack protection

Example of using subregions

You can use subregions for stack protection.

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 core to the disabled subregion causes the MPU to raise a background fault. The following figure shows an example of using subregions for stack protection.

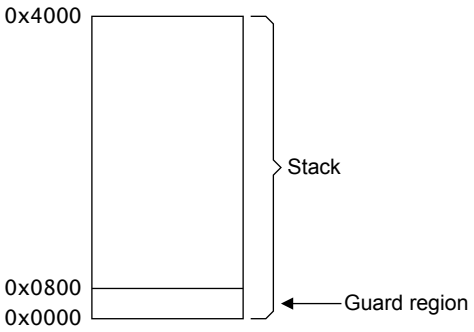


Figure 8-3 Overlapping subregion of memory

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 several physical memory areas sparsely distributed across the 4GB address space. If a programming error occurs, the core might issue an address that does not fall into any defined region.

If the address that the core 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, 15, 19, or 23 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 SCTLR. 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.

Related concepts

8.5 System configurability and QoS on page 8-159
8.6 Instruction and data TCM on page 8-161

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. in the following table 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 several cores share.
	Non-Shareable	-	For memory-mapped peripherals that only a single core uses.
Normal	Shareable	Non-Cacheable Write-Back Cacheable	For normal memory that is Shareable between several cores.
	Non-Shareable	Non-Cacheable Write-Back Cacheable	For normal memory that only a single core uses.

For more information on memory attributes and types, memory barriers, and ordering requirements for memory accesses, see the *Arm® Architecture Reference Manual Arm®v7-A and Arm®v7-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 the core is accessing Strongly-Ordered or Device type memory, its 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 several attributes associated with it. These control how a memory access is performed when the core accesses an address that falls within a given region.

The attributes are:

- Memory type, 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. in the following table 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 contain the Shareable bit, S. This bit usually determines whether the memory region is Shareable (`0b1`) or Non-Shareable (`0b0`). 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?
`0b000`	`0b0`	`0b0`	Strongly Ordered	Strongly Ordered	Shareable
`0b000`	`0b0`	`0b1`	Shareable Device	Device	Shareable
`0b000`	`0b1`	`0bX`	Reserved	-	-
`0b001`	`0b0`	`0b0`	Outer and Inner Non-Cacheable	Normal	S bit.[at]
`0b001`	`0b0`	`0b1`	Reserved	-	-
`0b001`	`0b1`	`0b0`			
`0b001`	`0b1`	`0b1`	Outer and Inner Write-Back, Write-Allocate	Normal	S bit[at]
`0b010`	`0b0`	`0b0`	Non-Shareable Device	Device	Non-Shareable
`0b010`	`0b0`	`0b1`	Reserved	-	-
`0b010`	`0b1`	`0bX`			
`0b011`	`0bX`	`0bX`			
`0b1BB`	`0bA`	`0bA`	Cacheable memory. See the Cacheable memory policies for the encoding for these bits. **`0bAA`** Inner policy **`0bBB`** Outer policy	Normal	S bit[at]

Cacheable memory policies

When TEX[2] == `0b1`, 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. in the following table shows the encoding.

[at] Region is Shareable if S == `0b1`, and Non-Shareable if S == `0b0`

Table 8-3 Inner and Outer cache policy encoding

Memory attribute encoding	Cache policy
0b00	Non-Cacheable
0b01	Write-Back, Write-Allocate
0b1X	Reserved

When the processor performs a memory access through its AXI3 bus master interface:

- The Inner attributes are indicated on the **AxUSERMx** signals.
- The Outer attributes are indicated on the **AxCACHEMx** signals.

For more information on region attributes, see the *Arm® Architecture Reference Manual Arm®v7-A and Arm®v7-R edition*.

Related concepts

8.2.2 Memory types on page 8-151

Related reference

MPU Region Access Control Registers on page 4-85

8.2.4 MPU interaction with memory system

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 core 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 Arm®v7-A and Arm®v7-R edition*.

8.2.5 MPU faults

The MPU can generate background faults, permission faults, and alignment faults. 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.

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.

Related reference

Region access permissions on page 8-149

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. Several instructions that access memory, for example, `LDM` and `STC`, require strict alignment.

See the *Arm® Architecture Reference Manual Arm®v7-A and Arm®v7-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.

Related reference

4.3.9 System Control Register on page 4-77

Related concepts

6.2 Fault handling on page 6-112

8.2.6 MPU software-accessible registers

The MPU memory region programming registers (CP15 registers) program the MPU regions.

Related concepts

8.2.4 MPU interaction with memory system on page 8-154

Related reference

4.3.12 MPU memory region programming registers on page 4-83

8.3 Branch prediction

Branch prediction uses both static and dynamic techniques. Dynamic branch prediction is used by default. But when there is no information history, static prediction is used instead.

Branch prediction predicts:

- 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 from the branch target 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 lookup/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 using the System Control Register. 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 low-latency interrupt mode, entry into an interrupt routine is slightly quicker, but with a slight reduction in global core 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.
- `VDIV` 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 core 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, the following table shows that Strongly-Ordered and Device read accesses, in addition to all store accesses, affect 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	Instructions per cycle performance	Level of determinism
Disabled	High	Medium
Enabled	High minus 3-4%	High

Related reference

4.3.9 System Control Register on page 4-77

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, data TCM, and the optional 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 an AXI fast peripheral port, AXI master port 1 when used with address filtering, the AXI low-latency peripheral port, or the data TCM.
- Any other transfers going through AXI master port 0 are considered to have low priority.

QoS is enabled by the QoS bit in the Auxiliary Control Register:

- 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. When 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 traffic 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 to the AXI fast peripheral ports.
- Peripherals connected on the AXI low-latency 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 core basis using ACTLR.QoS 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 when 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.

The following table 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 data TCM traffic only.	Do not set QoS bit	Do not set QoS bit
Peripheral and low-priority cacheable traffic, the data 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 data 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 recommended QoS bit settings 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 AXI low-latency peripheral port, an AXI fast peripheral port, or the data TCM.
- Other transfers, namely the cacheable transfers going through the AXI master port 0, have low priority.

Related reference

4.3.10 Auxiliary Control Register on page 4-80

8.6 Instruction and data TCM

Instruction and data TCMs are tightly-coupled in the Cortex-R8 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.

Note

- Arm recommends that the DTCM memory region is marked as XN in the MPU region settings to prevent instruction accesses to this address range.
- Arm recommends that the same double-word is not accessed at the same time by both the CPU and the AXI slave TCM port.

If an access corresponds to a TCM address, the access is treated as a cache hit and no other access is performed on the AXI buses.

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®-R8 MPCore Processor Integration Manual* for more information. The permissible TCM sizes are:

- 0KB.
- 4KB.
- 8KB.
- 16KB.
- 32KB.
- 64KB.
- 128KB.
- 256KB.
- 512KB.
- 1024KB.

Both TCMs can be preloaded using the AXI TCM slave port. This slave port provides access to the TCMs only.

From the view of a programmer:

- MPU regions targeting the TCM are private to the core 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 DTCM and ITCM Region Registers. 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.

If the processor is driven with **VINITHI[x]**=1 and **INITRAMx**=1 then the size of the ITCM is limited to a maximum size of 64KB.

The base address for each the TCM is configured using the relevant TCM Region register. The TCM base address must be aligned to the size of the TCM. The address ranges for the ITCM and DTCM should not overlap however if the address range for the ITCM and DTCM do overlap and both TCM instances are enabled then the DTCM will take precedence over the ITCM.

Both instruction and data TCM can be ECC protected.

Note

Write accesses to the instruction TCM are possible for debug purposes, but with limited throughput.

Related concepts

2.5.5 AXI TCM slave port on page 2-45

7.2.3 ECC on RAMs on page 7-132

Related reference

4.3.13 DTCM Region Register on page 4-88

4.3.14 ITCM Region Register on page 4-89

Chapter 9
Multiprocessing

This chapter describes the multiprocessing features of the Cortex-R8 processor, including the SCU.

It contains the following sections:

9.1 About multiprocessing and the SCU

The SCU connects Cortex-R8 processor cores to the memory system through the AXI3 interfaces.

The SCU functions are to:

- Maintain data cache coherency between the cores.
- Initiate L2 AXI3 memory accesses.
- Arbitrate between the cores requesting L2 accesses.
- Manage ACP accesses, with data cache coherency for the cores.

Note

The Cortex-R8 processor SCU does not support hardware management of instruction cache coherency.

The SCU has an optional *Accelerator Coherency Port* (ACP) that is used to connect a noncached master such as a DMA to the Cortex-R8 processor, as shown in the following figure.

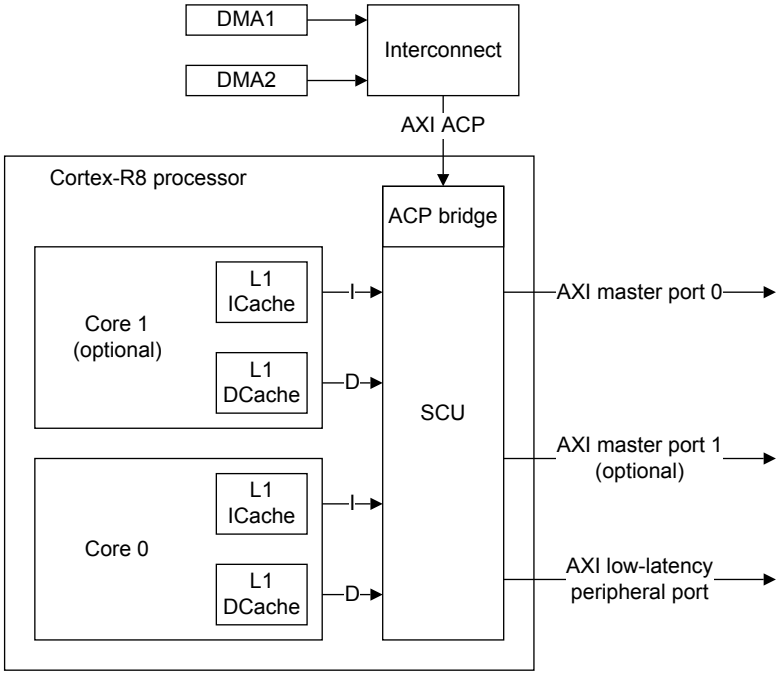


Figure 9-1 SCU and ACP in a two-core configuration

The SCU has two slave ports per core, one port connected to the instruction bus and one port connected to the data bus of each core.

The ACP is intended to be used for coherent data transfers. However, the ACP can also be used in a noncoherent mode, where any transfer with the level 2 memory can be done directly, without having to deal with data coherency.

The data coherency efficiency, either using the ACP or between individual cores, is enhanced by using the replicated tag RAMs of each L1 data cache tag RAM of each core in the SCU. 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 core without having to access that cache.

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.

If address filtering is not enabled, transfers for odd-numbered cores, and instructions such as SWP, LDREX/STREX, are routed through AXI master port 1, and transfers for even-numbered cores are routed through AXI master port 0.

The SCU also contains:

- An AXI memory-mapped low-latency peripheral port.
- Memory-mapped control registers.
- An interrupt controller.
- Private timers and watchdog.
- Global timers.

You can configure the event monitor for each core to gather statistics on the operation of the SCU.

Related concepts

2.5.2 AXI master port 1 on page 2-42

8.5 System configurability and QoS on page 8-159

2.5.3 AXI low-latency peripheral port on page 2-43

9.6 Global timer on page 9-207

Related reference

9.7.1 Coherent and noncoherent mode on page 9-211

9.3 SCU registers on page 9-167

9.4 Interrupt controller on page 9-189

9.5 Private timer and watchdog on page 9-201

10.1 Performance Monitoring Unit on page 10-215

9.2 Multiprocessing programmers view

To enable data cache coherency in the cores, set the SMP bit, bit[6] in the Auxiliary Control Register (the reset value is zero), and set the SCU enable bit, bit[0] in the SCU Control Register (the reset value is zero). The SCU enable bit must be set HIGH before any of the cores set their SMP bit HIGH.

The L1 data cache coherency between the cores is done in the inner Write-Back, Write-Allocate Shareable memory regions. The coherency between the ACP traffic and the L1 data cache of the cores is triggered when **AxUSERSC[0]** = 1 and **AxCACHESC[1]** = 1.

Related reference

4.3.10 Auxiliary Control Register on page 4-80
9.3.1 SCU Control Register on page 9-169
9.7.1 Coherent and noncoherent mode on page 9-211

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[11:0]** and **PFILTEREND[11:0]** define. The value of **PERIPHBASE[31:13]** can be retrieved by a core 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. The memory space in MB used for the address filtering is defined as follows:

The following table shows the peripheral accesses relative to the Peripheral Address setting.

Table 9-1 Peripheral accesses

Access	SCU registers		AXI low-latency peripheral port traffic
	Accessible by any core	Accessible through the ACP	Accessible by any core 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

The following table 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	*9.3.1 SCU Control Register* on page 9-169
0x04	SCU Configuration Register	IMPLEMENTATION DEFINED	*9.3.2 SCU Configuration Register* on page 9-171
0x08	SCU CPU Power Status Register	-	*9.3.3 SCU CPU Power Status Register* on page 9-173
0x0C	SCU Invalidate All Registers	0x0	*9.3.4 SCU Invalidate All Register* on page 9-175
0x40	Master Filtering Start Address Register	Defined by **MFILTERSTART** input	*9.3.5 Master Filtering Start Address Register* on page 9-176
0x44	Master Filtering End Address Register	Defined by **MFILTEREND** input	*9.3.6 Master Filtering End Address Register* on page 9-177
0x48	Peripherals Filtering Start Address Register	Defined by **PFILTERSTART** input	*9.3.7 LLP Filtering Start Address Register* on page 9-177

Table 9-2 SCU registers summary (continued)

Offset from PERIPHBASE[31:13]	Name	Reset value	Page
0x4C	Peripherals Filtering End Address Register	Defined by **PFILTEREND** input	*9.3.8 LLP Filtering End Address Register* on page 9-178
0x50	SCU Access Control Register	0b11	*9.3.9 SCU Access Control Register* on page 9-179
0x60	SCU Error Bank First Entry Register[au]	-	*9.3.10 SCU Error Bank First Entry Register* on page 9-180
0x64	SCU Error Bank Second Entry Register[au]	-	*9.3.11 SCU Error Bank Second Entry Register* on page 9-181
0x70	SCU Debug Tag RAM Operation Register	-	*SCU Debug Tag RAM Operation Register* on page 9-182
0x74	SCU Debug Tag RAM Data Value Register	-	*SCU Debug Tag RAM Data Value Register* on page 9-183
0x78	SCU Debug Tag RAM ECC Chunk Register[au]	-	*SCU Debug Tag RAM ECC Chunk Register* on page 9-184
0x7C	ECC Fatal Error Register[au]	0x0	*9.3.14 FPP Filtering Start Address Registers 0-3* on page 9-186
0x80	FPP Filtering Start Address Register for core 0	Defined by **FPFILTERSTART0**	*9.3.14 FPP Filtering Start Address Registers 0-3* on page 9-186
0x84	FPP Filtering End Address Register for core 0	Defined by **FPFILTEREND0**	*9.3.15 FPP Filtering End Address Registers 0-3* on page 9-187
0x88	FPP Filtering Start Address Register for core 1	Defined by **FPFILTERSTART1**	*9.3.14 FPP Filtering Start Address Registers 0-3* on page 9-186
0x8C	FPP Filtering End Address Register for core 1	Defined by **FPFILTEREND1**	*9.3.15 FPP Filtering End Address Registers 0-3* on page 9-187
0x90	FPP Filtering Start Address Register for core 2	Defined by **FPFILTERSTART2**	*9.3.14 FPP Filtering Start Address Registers 0-3* on page 9-186
0x94	FPP Filtering End Address Register for core 2	Defined by **FPFILTEREND2**	*9.3.15 FPP Filtering End Address Registers 0-3* on page 9-187
0x98	FPP Filtering Start Address Register for core 3	Defined by **FPFILTERSTART3**	*9.3.14 FPP Filtering Start Address Registers 0-3* on page 9-186
0x9C	FPP Filtering End Address Register for core 3	Defined by **FPFILTEREND3**	*9.3.15 FPP Filtering End Address Registers 0-3* on page 9-187

This section contains the following subsections:

[au] This register is present only when ECC is implemented.

9.3.1 SCU Control Register

The SCU Control Register enables speculative linefills to L2 with the L2C-310 Cache Controller.

The SCU Control Register also enables:

- 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.

The following figure shows the SCU Control Register bit assignments.

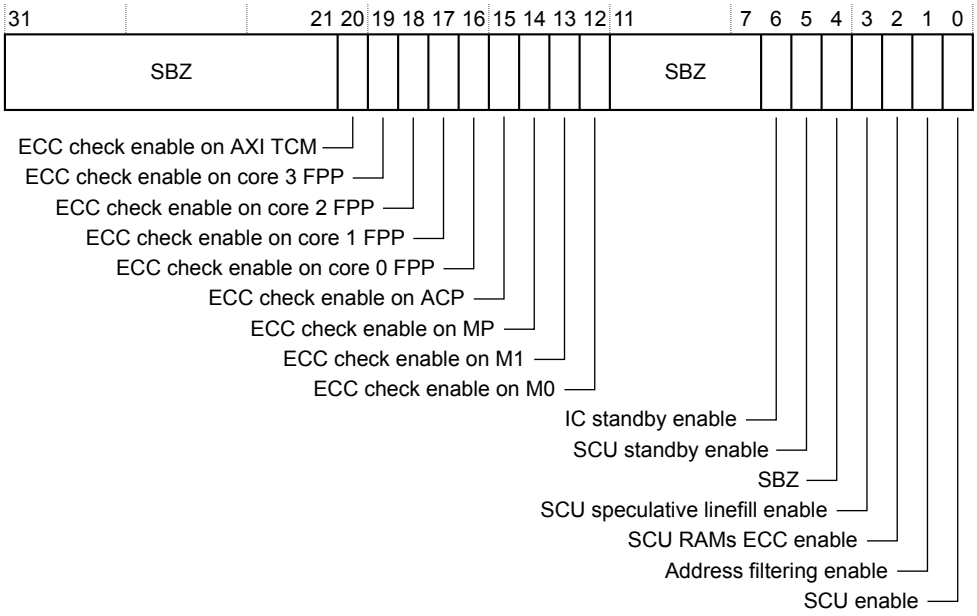


Figure 9-2 SCU Control Register bit assignments

The following table shows the SCU Control Register bit assignments.

Table 9-3 SCU Control Register bit assignments

Bits	Name	Description
[31:21]	-	Reserved. SBZ.
[20]	ECC check enable on AXI TCM slave port	When set, enables ECC check on the AXI TCM slave port.
[19]	ECC check enable on core 3 FPP	When set, enables ECC check on the AXI fast peripheral port for core 3.

Table 9-3 SCU Control Register bit assignments (continued)

Bits	Name	Description
[18]	ECC check enable on core 2 FPP	When set, enables ECC check on the AXI fast peripheral port for core 2.
[17]	ECC check enable on core 1 FPP	When set, enables ECC check on the AXI fast peripheral port for core 1.
[16]	ECC check enable on core 0 FPP	When set, enables ECC check on the AXI fast peripheral port for core 0.
[15]	ECC check enable on ACP	When set, enables ECC check on the *Accelerator Coherency Port* (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 no core is performing a read/write request.
[5]	SCU standby enable	When set, the clock in the SCU is turned off when all cores 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, **ARREADYSC**, **AWREADYSC**, and **WREADYSC** on the ACP are forced LOW. The clock is turned on when any core leaves WFI mode, or if there is a new request on the ACP.
[4]	-	Reserved. SBZ.
[3]	SCU speculative linefill enable	When set, coherent linefill requests are sent speculatively to the L2C-310 Cache Controller in parallel with the tag lookup. If the tag lookup misses, the confirmed linefill is sent to the L2C-310 Cache Controller and receives RDATA earlier because the speculative request already initiated the data 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: `0b1` ECC on. `0b0` ECC off. This is the default setting. This bit is always zero if support for ECC is not implemented.

Table 9-3 SCU Control Register bit assignments (continued)

Bits	Name	Description
[1]	Address filtering enable	This is a read-only bit that indicates address filtering: `0b1` Address filtering on. `0b0` Address filtering off. This value is the value of **MFILTEREN** sampled when **nSCURESET** is deasserted. This bit is always zero if the SCU is implemented in the single master port configuration. See *2.5.2 AXI master port 1* on page 2-42.
[0]	SCU enable	Enables SCU: `0b1` SCU enabled. `0b0` SCU disabled. This is the default setting.

9.3.2 SCU Configuration Register

The SCU Configuration Register reads tag RAM sizes for the cores that are present, determines the cores that are taking part in coherency, and reads the number of cores present.

Usage constraints
This register is read-only.

Configurations
Available in all configurations.

Attributes

Offset from **PERIPHBASE[31:13]**: `0x04`

Reset value: IMPLEMENTATION DEFINED

The following figure shows the SCU Configuration Register bit assignments.

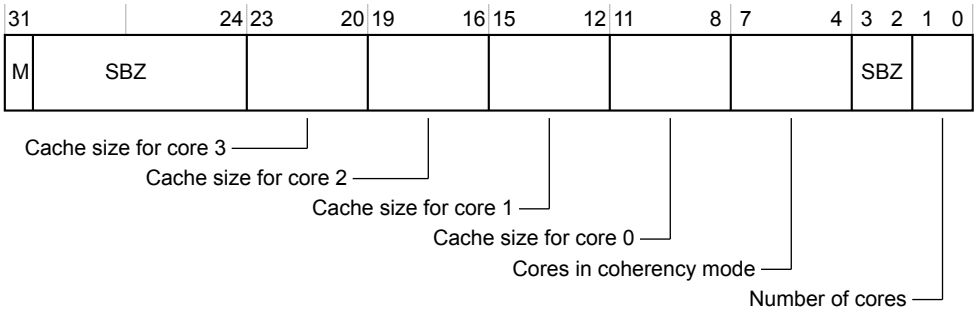


Figure 9-3 SCU Configuration Register bit assignments

The following table shows the SCU Configuration Register bit assignments.

Table 9-4 SCU Configuration Register bit assignments

Bits	Name	Description
[31]	M	This read-only bit indicates the inclusion of the AXI master port 1: `0b0` AXI master port 1 not present. `0b1` AXI master port 1 present.
[30:24]	-	Reserved. SBZ.

Table 9-4 SCU Configuration Register bit assignments (continued)

Bits	Name	Description
[23:20]	Cache size for core 3	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.
[19:16]	Cache size for core 2	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.
[15:12]	Cache size for core 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 core 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.

Table 9-4 SCU Configuration Register bit assignments (continued)

Bits	Name	Description
[7:4]	Cores in coherency mode	Shows the cores that are in *Symmetric Multiprocessing* (SMP) or *Asymmetric Multiprocessing* (AMP) mode: `0b1` This core is in SMP mode taking part in coherency. `0b0` This core is in AMP mode not taking part in coherency or not present. **[7]** Core 3. **[6]** Core 2. **[5]** Core 1. **[4]** Core 0.
[3:2]	-	Reserved. SBZ.
[1:0]	Number of cores	Number of cores present in the Cortex-R8 processor: `0b11` Four cores, core 0, core 1, core 2 and core 3. `0b10` Three cores, core 0, core 1 and core 2. `0b01` Two cores, core 0 and core 1. `0b00` One core, core 0.

Related reference
9.3 SCU registers on page 9-167

9.3.3 SCU CPU Power Status Register

The SCU CPU Power Status Register specifies the state of the Cortex-R8 processor cores with reference to power modes.

Usage constraints

Writes to this register are enabled when the access bit for the core is set in the SCU Access Control Register. See *9.3.9 SCU Access Control Register* on page 9-179.

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 core must set its status field to signal to the power controller the mode it is about to enter. The core power down routine must then execute a DSB instruction and then a WFI entry instruction. When in WFI state, the **PWRCTLOx** bus is enabled and signals to the power controller what it must do with power domains. See also *2.4.1 Individual core power management* on page 2-36.

The SCU CPU Power Status Register bits can also be read by a core exiting low-power mode to determine its state before executing its reset setup.

Configurations

Available in all configurations.

Attributes

Offset from **PERIPHBASE[31:13]**: `0x08`

Reset value: -

The following figure shows the SCU CPU Power Status Register bit assignments.

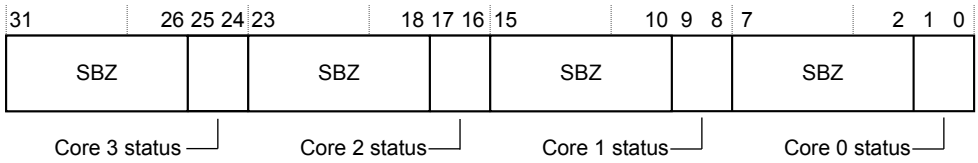


Figure 9-4 SCU CPU Power Status Register bit assignments

The following table shows the SCU CPU Power Status Register bit assignments.

Table 9-5 SCU CPU Power Status Register bit assignments

Bits	Name	Description
[31:26]	-	Reserved. SBZ.
[25:24]	Core 3 status	Power status of core 3: **`0b0X`** Core must be powered on. **`0b10`** Core can enter dormant mode. **`0b11`** Core can enter powered-off mode.
[23:18]	-	Reserved. SBZ.
[17:16]	Core 2 status	Power status of core 2: **`0b0X`** Core must be powered on. **`0b10`** Core can enter dormant mode. **`0b11`** Core can enter powered-off mode.
[15:10]	-	Reserved. SBZ.
[9:8]	Core 1 status	Power status of core 1: **`0b0X`** Core must be powered on. **`0b10`** Core can enter dormant mode. **`0b11`** Core can enter powered-off mode.

Table 9-5 SCU CPU Power Status Register bit assignments (continued)

Bits	Name	Description
[7:2]	-	Reserved. SBZ.
[1:0]	Core 0 status	Power status of core 0: **0b0X** Core must be powered on. **0b10** Core can enter dormant mode. **0b11** Core can enter powered-off mode.

Related concepts

2.4.1 Individual core power management on page 2-36

Related reference

9.3.9 SCU Access Control Register on page 9-179

9.3 SCU registers on page 9-167

9.3.4 SCU Invalidate All Register

The SCU Invalidate All Register invalidates the SCU tag RAMs on a per core, and per way basis.

Usage constraints

- This register invalidates all lines in the selected ways.
- Writes to this register are enabled when the access bit for the core is set in the SCU Access Control Register. See *9.3.9 SCU Access Control Register* on page 9-179.

Configurations

Available in all configurations.

Attributes

Offset from **PERIPHBASE[31:13]**: `0x0C`

Reset value: 0x0

The following figure shows the SCU Invalidate All Register bit assignments.

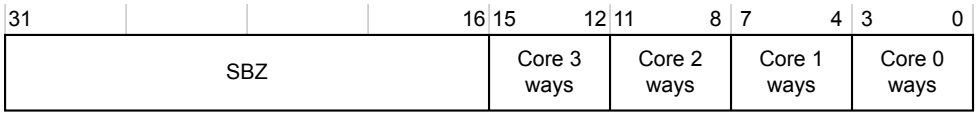


Figure 9-5 SCU Invalidate All Register bit assignments

The following table shows the SCU Invalidate All Register bit assignments.

Table 9-6 SCU Invalidate All Register bit assignments

Bits	Name	Description
[31:16]		Reserved. SBZ.
[15:12]	Core 3 ways	Specifies the ways that must be invalidated for core 3. Writing to these bits has no effect if the Cortex-R8 processor has fewer than four cores.
[11:8]	Core 2 ways	Specifies the ways that must be invalidated for core 2. Writing to these bits has no effect if the Cortex-R8 processor has fewer than three cores.

Table 9-6 SCU Invalidate All Register bit assignments (continued)

Bits	Name	Description
[7:4]	Core 1 ways	Specifies the ways that must be invalidated for core 1. Writing to these bits has no effect if the Cortex-R8 processor has fewer than two cores.
[3:0]	Core 0 ways	Specifies the ways that must be invalidated for core 0.

Related reference

9.3.9 SCU Access Control Register on page 9-179

9.3 SCU registers on page 9-167

9.3.5 Master Filtering Start Address Register

The Master Filtering Start Address Register provides the start address for use with master port 1 in a two-master port configuration.

Usage constraints

This register is read-only.

Configurations

Available in all two-master port configurations. When only one master port is present, these registers are not implemented. Writes have no effect and reads return a value `0x0` for all filtering registers.

Attributes

Offset from **PERIPHBASE[31:13]**: `0x40`

Reset value: Defined by MFILTERSTART input.

The following figure shows the Master Filtering Start Address Register bit assignments.

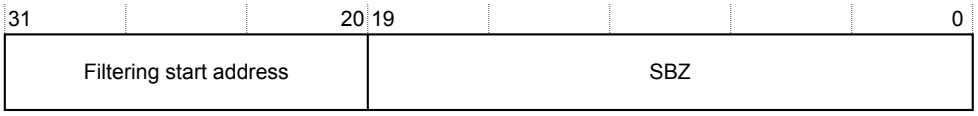


Figure 9-6 Master Filtering Start Address Register bit assignments

The following table shows the Master Filtering Start Address Register bit assignments.

Table 9-7 Master Filtering Start Address Register bit assignments

Bits	Name	Description
[31:20]	Filtering start address	Start address for use with master port 1 in a two-master port configuration when address filtering is enabled. This value is the value of **MFILTERSTART** sampled on exit from reset. The value on the input gives the upper address bits with 1MB granularity.
[19:0]		Reserved. SBZ.

See *2.5.2 AXI master port 1* on page 2-42.

Related concepts

2.5.2 AXI master port 1 on page 2-42

Related reference

9.3 SCU registers on page 9-167

9.3.6 Master Filtering End Address Register

The Master Filtering End Address Register provides the end address for use with master port 1 in a two-master port configuration.

Usage constraints

This register 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.

Configurations

Available in all two-master port configurations. When only one master port is present writes have no effect and reads return a value `0x0` for all filtering registers.

Attributes

Offset from **PERIPHBASE[31:13]**: `0x44`

Reset value: Defined by MFILTEREND input.

The following figure shows the Master Filtering End Address Register bit assignments.

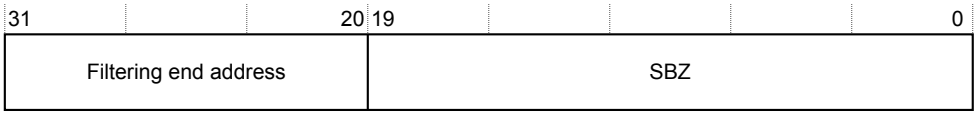


Figure 9-7 Master Filtering End Address Register bit assignments

The following table 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 **MFILTEREND** sampled on exit from reset. The value on the input gives the upper address bits with 1MB granularity.
[19:0]		Reserved. SBZ.

See *A.5 Configuration signals* on page Appx-A-362. See also *2.5.2 AXI master port 1* on page 2-42.

Related concepts

2.5.2 AXI master port 1 on page 2-42

Related reference

9.3 SCU registers on page 9-167
A.5 Configuration signals on page Appx-A-362

9.3.7 LLP Filtering Start Address Register

The LLP Filtering Start Address Register provides the filtering start address for the AXI low-latency peripheral port.

Usage constraints

This register is read-only. For the peripheral port region to operate, the filtering start address must be lower than the filtering end address.

Configurations

Available in all configurations.

Attributes

Offset from **PERIPHBASE[31:13]**: 0x48

Reset value: Defined by PFILTERSTART input.

The following figure shows the LLP Filtering Start Address Register bit assignments.

Figure 9-8 LLP Filtering Start Address Register bit assignments

The following table shows the LLP Filtering Start Address Register bit assignments.

Table 9-9 LLP 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 **PFILTERSTART** sampled on exit from reset. The value on the input gives the upper address bits with 1MB granularity.
[19:0]		Reserved. SBZ.

See *2.5.3 AXI low-latency peripheral port* on page 2-43.

Related concepts

2.5.3 AXI low-latency peripheral port on page 2-43

Related reference

9.3 SCU registers on page 9-167

9.3.8 LLP Filtering End Address Register

The LLP Filtering End Address Register provides the filtering end address for the AXI low-latency peripheral port.

Usage constraints

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.

Configurations

Available in all configurations.

Attributes

Offset from **PERIPHBASE[31:13]**: 0x4C

Reset value: Defined by PFILTEREND input.

The following figure shows the LLP Filtering End Address Register bit assignments.

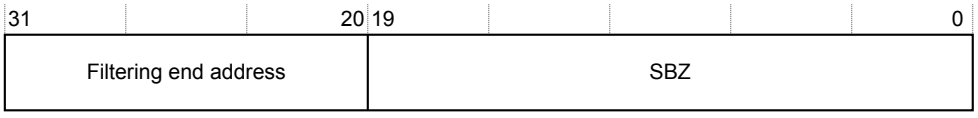


Figure 9-9 LLP Filtering End Address Register bit assignments

The following table shows the LLP Filtering End Address Register bit assignments.

Table 9-10 LLP 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 **PFILTEREND** sampled on exit from reset. The value on the input gives the upper address bits with 1MB granularity.
[19:0]		Reserved. SBZ.

See *A.5 Configuration signals* on page Appx-A-362. See also *2.5.3 AXI low-latency peripheral port* on page 2-43.

Related concepts

2.5.3 AXI low-latency peripheral port on page 2-43

Related reference

9.3 SCU registers on page 9-167

A.5 Configuration signals on page Appx-A-362

9.3.9 SCU Access Control Register

The SCU Access Control Register controls access to SCU registers on a per core basis.

The SCU Access Control Register controls access to the following registers:

- SCU Control Register.
- SCU CPU Power Status Register.
- SCU Invalidate All Register.
- SCU Error Bank First Entry Register.
- SCU Error Bank Second Entry Register.

The following figure shows the SCU Access Control Register bit assignments.

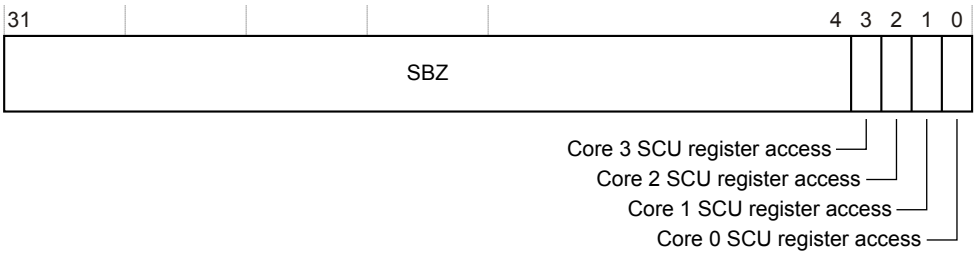


Figure 9-10 SCU Access Control Register bit assignments

The following table shows the SCU Access Control Register bit assignments.

Table 9-11 SCU Access Control Register bit assignments

Bits	Name	Description	
[31:4]		Reserved. SBZ.	
[3]	Core 3 SCU register access	`0b1`	Core 3 can access the SCU registers. This is the default.
		`0b0`	Core 3 cannot access the SCU registers.
[2]	Core 2 SCU register access	`0b1`	Core 2 can access the SCU registers. This is the default.
		`0b0`	Core 2 cannot access the SCU registers.

Table 9-11 SCU Access Control Register bit assignments (continued)

Bits	Name	Description	
[1]	Core 1 SCU register access	`0b1`	Core 1 can access the SCU registers. This is the default.
		`0b0`	Core 1 cannot access the SCU registers.
[0]	Core 0 SCU register access	`0b1`	Core 0 can access the SCU registers. This is the default.
		`0b0`	Core 0 cannot access the SCU registers.

Related reference

9.3.1 SCU Control Register on page 9-169
9.3.3 SCU CPU Power Status Register on page 9-173
9.3.4 SCU Invalidate All Register on page 9-175
9.3.10 SCU Error Bank First Entry Register on page 9-180
9.3.11 SCU Error Bank Second Entry Register on page 9-181

9.3.10 SCU Error Bank First Entry Register

The SCU Error Bank First Entry Register shows the first SCU error bank entry.

Usage constraints

There are no usage constraints.

Configurations

Available only in configurations where ECC is implemented.

Attributes

Offset from **PERIPHBASE[31:13]**: `0x60`

Reset value: -

The following figure shows the SCU Error Bank First Entry Register bit assignments.

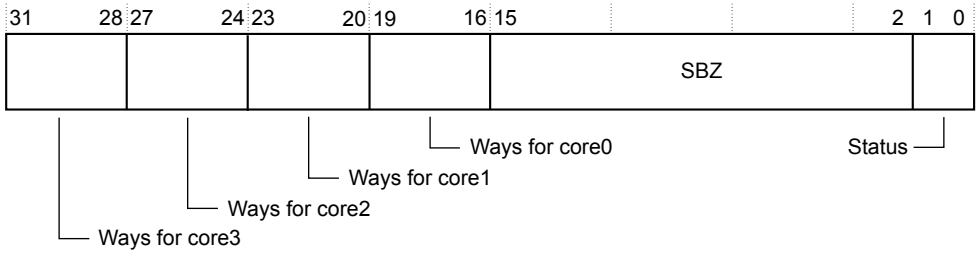


Figure 9-11 SCU Error Bank First Entry Register bit assignments

The following table shows the SCU Error Bank First Entry Register bit assignments.

Table 9-12 SCU Error Bank First Entry Register bit assignments

Bits	Name	Function
[31:28]	Ways for core3	Ways for core 3
[27:24]	Ways for core2	Ways for core 2.
[23:20]	Ways for core1	Ways for core 1.
[19:16]	Ways for core0	Ways for core 0.

Table 9-12 SCU Error Bank First Entry Register bit assignments (continued)

Bits	Name	Function
[15:2]	-	Reserved. SBZ.
[1:0]	Status	Error status. The values are: `0b00` No error. `0b01` Error seen by SCU tag RAM, but not handled by core. `0b10` Error seen by both SCU and core. `0b11` Error is confirmed by software.

Related reference

9.3 SCU registers on page 9-167

9.3.11 SCU Error Bank Second Entry Register

The SCU Error Bank Second Entry Register shows the second SCU error bank entry.

Usage constraints

There are no usage constraints.

Configurations

Available only in configurations where ECC is implemented.

Attributes

Offset from **PERIPHBASE[31:13]**: `0x64`

Reset value: -

The following figure shows the SCU Error Bank Second Entry Register bit assignments.

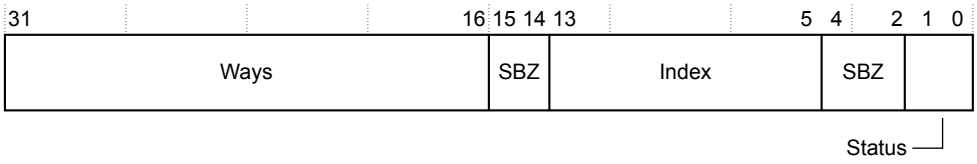


Figure 9-12 SCU Error Bank Second Entry Register bit assignments

The following table shows the SCU Error Bank Second Entry Register bit assignments.

Table 9-13 SCU Error Bank Second Entry Register bit assignments

Bits	Name	Function
[31:16]	Ways	Ways in the SCU tag RAM, four bits per core.
[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 core. `0b10` Error seen by both SCU and core. `0b11` Error is confirmed by software.

Related reference

9.3 SCU registers on page 9-167

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 to select a specific tag value.
- SCU Debug Tag RAM ECC Chunk Register to select the ECC chunk associated with the tag value.

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 or the SCU Debug Tag RAM ECC Chunk Register.

To write a given SCU tag RAM location:

1. Write the SCU Debug Tag RAM Data Value Register or the SCU Debug Tag RAM ECC Chunk Register.
2. Write the SCU Debug Tag RAM Operation Register.

SCU Debug Tag RAM Operation Register

The SCU Debug Tag RAM Operation Register gives the address and action for SCU tag RAM direct access.

Usage constraints

There are no usage constraints.

Configurations

Available in all configurations.

Attributes

Offset from **PERIPHBASE[31:13]**: `0x70`

Reset value: -

The following figure shows the SCU Debug Tag RAM Operation Register bit assignments.

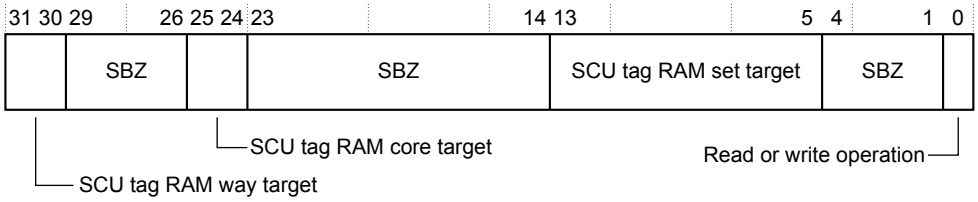


Figure 9-13 SCU Debug Tag RAM Operation Register bit assignments

The following table 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.
[29:26]	-	Reserved. SBZ.

Table 9-14 SCU Debug Tag RAM Operation Register bit assignments (continued)

Bits	Name	Function
[25:24]	SCU tag RAM core target	Indicates the core target: `0b00` Core 0. `0b01` Core 1. `0b10` Core 2. `0b11` Core 3.
[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: `0b0` Read. `0b1` Write.

Related reference
9.3 SCU registers on page 9-167

SCU Debug Tag RAM Data Value Register

The SCU Debug Tag RAM Data Value Register gives the data value for SCU tag RAM direct access.

Usage constraints
There are no usage constraints.

Configurations
Available in all configurations.

Attributes

Offset from **PERIPHBASE[31:13]**: `0x74`

Reset value: -

The following figure shows the SCU Debug Tag RAM Data Value Register bit assignments.

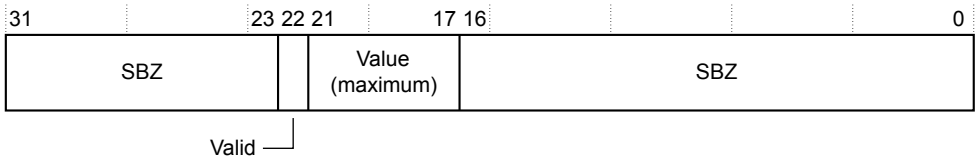


Figure 9-14 SCU Debug Tag RAM Data Value Register bit assignments

The following table 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.

Table 9-15 SCU Debug Tag RAM Data Value Register bit assignments (continued)

Bits	Name	Function
[21:17]	Value	Tag value: **[21]** 4KB. **[21:20]** 8KB. **[21:19]** 16KB. **[21:18]** 32KB. **[21:17]** 64KB. Unused bits are Reserved.
[16:0]	-	Reserved. SBZ.

Related reference
9.3 SCU registers on page 9-167

SCU Debug Tag RAM ECC Chunk Register

The SCU Debug Tag RAM ECC Chunk Register shows the ECC chunk value.

Usage constraints
There are no usage constraints.

Configurations
Available only in configurations where ECC is implemented.

Attributes

Offset from **PERIPHBASE[31:13]**: `0x78`

Reset value: -

The following figure shows the SCU Debug Tag RAM ECC Chunk Register bit assignments.

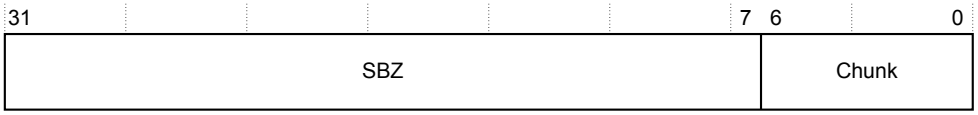


Figure 9-15 SCU Debug Tag RAM ECC Chunk Register bit assignments

The following table 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]	-	Reserved. SBZ.
[6:0]	Chunk	ECC chunk value.

Related reference
9.3 SCU registers on page 9-167

Related reference

SCU Debug Tag RAM Operation Register on page 9-182
SCU Debug Tag RAM Data Value Register on page 9-183
SCU Debug Tag RAM ECC Chunk Register on page 9-184

9.3.13 ECC Fatal Error Register

The ECC Fatal Error Register provides a double-bit ECC fatal error signal that indicates data written out of the processor might be corrupted. The system can use this signal to disable writes to external memory.

Usage constraints

Writes to this register are enabled when the access bit for the core is set in the SCU Access Control Register, see *9.3.9 SCU Access Control Register* on page 9-179.

Configurations

Available only in configurations where ECC is implemented.

Attributes

Offset from **PERIPHBASE[31:13]**: `0x7C`

Reset value: `0x0`

The following figure shows the ECC Fatal Error Register bit assignments.

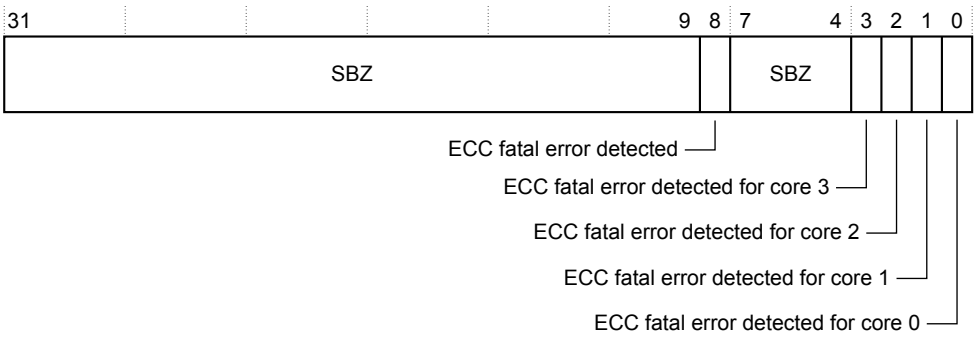


Figure 9-16 ECC Fatal Error Register bit assignments

Table 9-17 ECC Fatal Error Register bit assignments

Bits	Name	Function
[31:9]	-	Reserved. SBZ.
[8]	ECC fatal error detected	Set only when any core has an ECC fatal error. Reset by software. This bit is exported externally by **FATALERRDET**.
[7:4]	-	Reserved. SBZ.
[3]	ECC fatal error detected for core 3	Set only when core 3 has an ECC fatal error. Reset by software.
[2]	ECC fatal error detected for core 2	Set only when core 2 has an ECC fatal error. Reset by software.

Table 9-17 ECC Fatal Error Register bit assignments (continued)

Bits	Name	Function
[1]	ECC fatal error detected for core 1	Set only when core 1 has an ECC fatal error. Reset by software.
[0]	ECC fatal error detected for core 0	Set only when core 0 has an ECC fatal error. Reset by software.

Related reference

9.3.9 SCU Access Control Register on page 9-179

9.3 SCU registers on page 9-167

9.3.14 FPP Filtering Start Address Registers 0-3

The FPP Filtering Start Address Registers provide the filtering start address of the FPP that corresponds to the core.

Usage constraints

These registers are read-only. The filtering start address must be lower than the filtering end address of the FPP that corresponds to the core.

Configurations

Available when the FPP of the corresponding core is implemented.

Attributes

Offset from **PERIPHBASE[31:13]**:

- Core 0: `0x80`
- `Core 1: 0x88`
- `Core 2: 0x90`
- `Core 3: 0x98`

Reset value: Defined by FPFILTERSTARTx.

The following figure shows the FPP Filtering Start Address Registers 0-3 bit assignments.

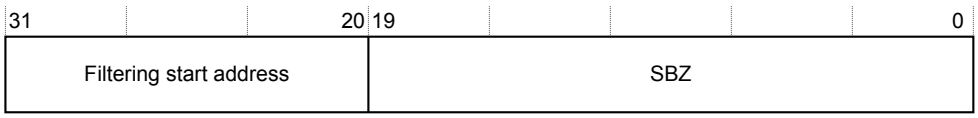


Figure 9-17 FPP Filtering Start Address Registers 0-3

The following table shows the FPP Filtering Start Address Registers 0-3 bit assignments.

Table 9-18 FPP Filtering Start Address Registers 0-3 bit assignments

Bits	Name	Function
[31:20]	Filtering start address	Filtering start address for core 0-3. The default value is the value of **FPFILTERSTARTx**, where **x** corresponds to either core 0, 1, 2 or 3, sampled when the core exits reset. The value on the input gives the upper address bits with 1MB granularity.
[19:0]	-	Reserved. SBZ.

Note

The filtering start address is applied independently of the AXI request type and its attributes.

Related reference

9.3 SCU registers on page 9-167

9.3.15 FPP Filtering End Address Registers 0-3

The FPP Filtering End Address Registers provide the filtering end address of the FPP that corresponds to the core.

Usage constraints

These registers are read-only. The filtering end address must be higher than the filtering start address of the FPP that corresponds to the core. These registers have an inclusive address as their end address. This means that the topmost megabyte of memory address space can be included in the filtering address range.

Configurations

Available when the FPP of the corresponding core is implemented.

Attributes

Offset from PERIPHBASE[31:13]:

- Core 0: `0x84`
- `Core 1: 0x8C`
- `Core 2: 0x94`
- `Core 3: 0x9C`

Reset value: Defined by FPFILTERENDx.

The following figure shows the FPP Filtering End Address Registers 0-3 bit assignments.

Figure 9-18 FPP Filtering End Address Registers 0-3

The following table shows the FPP Filtering End Address Registers 0-3 bit assignments.

Table 9-19 FPP Filtering End Address Registers 0-3 bit assignments

Bits	Name	Function
[31:20]	Filtering end address	Filtering end address for core 0-3. The default value is the value of **FPFILTERENDx**, where **x** corresponds to either core 0, 1, 2 or 3, sampled when the core exits reset. The value on the input gives the upper address bits with 1MB granularity.
[19:0]	-	Reserved. SBZ.

Note

The filtering end address is applied independently of the AXI request type and its attributes.

Related reference

9.3 SCU registers on page 9-167

Related reference

9.3.7 LLP Filtering Start Address Register on page 9-177

9.4 Interrupt controller

The interrupt controller is a single functional unit that is located in a Cortex-R8 processor design. There is one interrupt interface per core in the design. This implementation of the interrupt controller does not support the Security Extensions.

Interrupts controlled by the interrupt controller are only signaled as IRQ exceptions. Only the legacy nFIQ inputs can signal an FIQ exception.

The interrupt controller is memory-mapped. The Cortex-R8 processor cores access it by using a private interface through the SCU.

The interrupt controller is compliant with the Arm *Generic Interrupt Controller* (GIC) v1.0 architecture. The information in this TRM 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 contains the following subsections:

9.4.1 Interrupt controller clock frequency

The interrupt controller runs on **PERIPHCLK**. The clock period is configured, during integration, as an integer division of the Cortex-R8 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.

The timers and watchdogs use the same clock as the interrupt controller.

Related concepts

2.3 Clocking, resets, and initialization on page 2-29

9.4.2 Interrupt distributor interrupt sources

The interrupt distributor centralizes all interrupt sources before dispatching the highest priority ones to each Cortex-R8 processor core.

The Cortex-R8 processor supports only the 1-N service model for SPIs.

All interrupt sources are identified by a unique ID. All interrupt sources have their own configurable priority and list of targeted Cortex-R8 processor cores. If two interrupts have the same programmed priority level, then the interrupt with the lower ID value will take priority over the interrupt with the higher ID. This is a list of cores that the interrupt is sent to when triggered by the interrupt distributor.

Interrupt sources are of the following types:

Software Generated Interrupts (SGI)
Each core 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 core to determine its own core ID when it deals with SGIs. The priority of an SGI depends on the value set by the receiving core in the banked SGI priority registers, not the priority set by the sending core.

Global timer, PPI[0]
The global timer uses ID27.

A legacy nFIQ input, PPI[1]

In the Cortex-R8 processor, the **nFIQ** input is connected both to the corresponding core, where it causes an FIQ exception, and to the interrupt controller. If the interrupt controller is enabled, then **nFIQ** is mapped to PPI[1], and can also cause an IRQ exception.

The **nFIQ** input can be used to generate either FIQ exceptions or IRQ exceptions. You must configure the interrupt controller to ensure that it does not generate both IRQ and FIQ exceptions. To use the **nFIQ** input to generate FIQ exceptions, you must either disable interrupt ID28 in the interrupt controller or disable the interrupt controller itself.

To use the **nFIQ** input to generate IRQ exceptions, you enable interrupt ID28 in the interrupt controller and you should mask FIQ exceptions by setting bit[6] of the CPSR, the F bit, so that IRQ exceptions are used.

Private timer, PPI[2]

Each core has its own private timers that can generate interrupts, using ID29.

Watchdog timers, PPI[3]

Each core has its own watchdog timers that can generate interrupts, using ID30.

A legacy nIRQ input, PPI[4]

In legacy IRQ mode the legacy **nIRQ** input, on a per-core basis, bypasses the interrupt distributor logic and directly drives interrupt requests into the core.

When a core uses the interrupt controller, rather than the legacy input in the legacy mode, by enabling its own core 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 (rising edge) or level sensitive (HIGH level). SPIs start at ID32. The **IRQS** bus generates SPIs.

Note

The Cortex-R8 processor does not provide internal synchronization for the interrupt signals, **nIRQ**, **nFIQ**, and **IRQS**.

Related concepts

9.6 Global timer on page 9-207

Related reference

9.5 Private timer and watchdog on page 9-201

9.4.3 Priority formats

The processor implements a four-bit version of the Arm Interrupt Controller Architecture Specification.

See the *Arm® Generic Interrupt Controller Architecture Specification*.

9.4.4 Distributor register descriptions

Summary of the distributor register.

Registers not described in the distributor register summary table are RAZ/WI. The information in the table does not reproduce information about registers already described in the *Arm® Generic Interrupt Controller Architecture Specification*.

The ICDIPR and ICDIPTR registers are byte accessible and word accessible. All other registers in the table are word accessible. Any other access is UNPREDICTABLE.

Distributor register summary table

The offset of this page from **PERIPHBASE[31:13]** is `0x1000-0x1FFF`.

Table 9-20 Distributor register summary

Base	Name	Type	Reset	Width	Description
`0x000`	ICDDCR	RW	`0x00000000`	32	*Distributor Control Register* on page 9-191
`0x004`	ICDICTR	RO	Configuration dependent	32	*Interrupt Controller Type Register* on page 9-192
`0x008`	ICDIIDR	RO	`0x0400043B`	32	*Distributor Implementer Identification Register* on page 9-194
`0x00C-0x09C`	-	-	-	-	Reserved
`0x100-0x13C`	ICDISERn	RW	`0x00000000`[av]	32	Interrupt Set-Enable Registers
`0x180-0x1BC`	ICDICERn	RW	`0x00000000`[aw]	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[ax]
`0x7FC`	-	-	-	-	Reserved
`0x800-0x9FC`	ICDIPTRn	RW[aw]	`0x0000000`	32	Interrupt Core Targets Registers
`0xBFC`	-	-	-	-	Reserved
`0xC00-0xC7C`	ICDICFRn	RW	Implementation dependent	32	Interrupt Configuration Registers
`0xD00`	PPI Status	-	`0x00000000`	32	*PPI Status Register* on page 9-195
`0xD04-0xD3C`	SPI Status	RO	`0x00000000`	32	*SPI Status Registers* on page 9-196
`0xD80-0xEFC`	-	-	-	-	Reserved
`0xF00`	ICDSGIR	WO	-	32	Software Generated Interrupt Register
`0xF04-0xFCC`	-	-	-	-	Reserved
`0xFD0-0xFEC`	Peripheral Identification [4:0]	RO	Configuration dependent	8	*Identification registers* on page 9-198
`0xFF0-0xFFC`	Component Identification [3:0]	RO	-	8	

Distributor Control Register

The ICDDCR controls whether the distributor responds to external stimulus changes that occur on SPI and PPI signals.

Usage constraints

There are no usage constraints.

av The reset value for the registers that contain the SGI and PPI interrupts is IMPLEMENTATION DEFINED.
aw Not configurable. Reset to 1.
ax Only the top four bits of each 8-bit field of the register are in use.

Configurations

Available in all configurations.

Attributes

Base: `0x000`

Name: ICDDCR

Type: RW

Reset: `0x00000000`

Width: 32

The following figure shows the ICDDCR bit assignments.

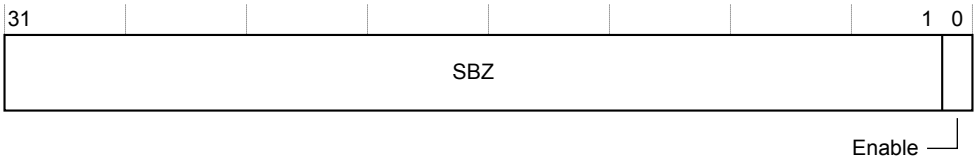


Figure 9-19 ICDDCR bit assignments

The following table shows the ICDDCR bit assignments.

Table 9-21 ICDDCR bit assignments

Bits	Name	Function
[31:1]	-	Reserved. SBZ.
[0]	Enable	The encoding is: `0b0` Disables all interrupt control bits in the distributor from changing state because of any external stimulus change that occurs on the corresponding SPI or PPI signals. `0b1` Enables the distributor to update register locations for interrupts.

Related reference

Distributor register summary table on page 9-191

Interrupt Controller Type Register

The ICDICTR provides information about the configuration of the interrupt controller.

Usage constraints

There are no usage constraints.

Configurations

Available in all configurations.

Attributes

Base: `0x004`

Name: ICDICTR

Type: RO

Reset: Configurationdependent

Width: 32

The following figure shows the ICDICTR bit assignments.

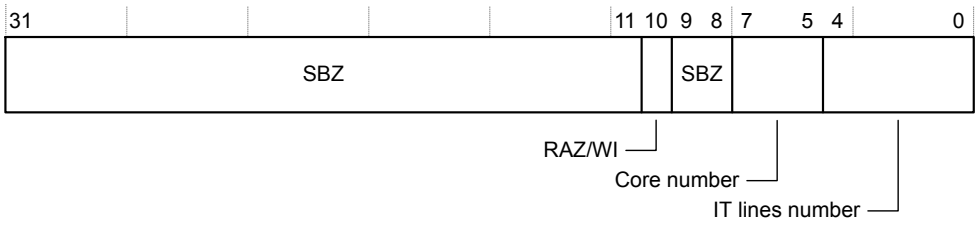


Figure 9-20 ICDICTR bit assignments

The following table shows the ICDICTR bit assignments.

Table 9-22 ICDICTR bit assignments

Bits	Name	Function
[31:11]		Reserved. SBZ.
[10]	-	Reserved. RAZ/WI.
[9:8]	-	Reserved. SBZ.
[7:5]	Core number	The encoding is: `0b000` The configuration contains one core. `0b001` The configuration contains two cores. `0b010` The configuration contains three cores. `0b011` The configuration contains four cores. All other values are unused.
[4:0]	IT lines number	The encoding is: `0b00000` The distributor provides 32 interrupts[ay], 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.

Related reference

Distributor register summary table on page 9-191

[ay] The distributor always uses interrupts of IDs 0-31 to control any SGIs and PPIs that the interrupt controller might contain.

Interrupt Core Targets Registers

For systems that support only one core, all ICDIPTRn registers read as zero, and writes are ignored.

Note

If the Processor Target field is set to `0b0` 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

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 ICDIIDR provides information about the implementer and the revision of the controller.

Usage constraints

There are no usage constraints.

Configurations

Available in all configurations.

Attributes

Base: `0x008`

Name: ICDIIDR

Type: RO

Reset: `0x0400043B`

Width: 32

The following figure shows the ICDIIDR bit assignments.

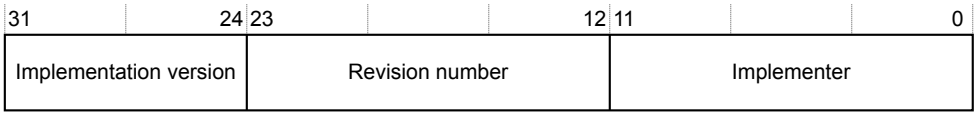


Figure 9-21 ICDIIDR bit assignments

The following table shows the ICDIIDR bit assignments.

Table 9-23 ICDIIDR bit assignments

Bits	Values	Name	Description
[31:24]	`0x04`	Implementation version	Gives implementation version number
[23:12]	`0x00`	Revision number	Returns the revision number of the controller
[11:0]	`0x43B`	Implementer	Implementer number

Related reference

Distributor register summary table on page 9-191

PPI Status Register

The PPI Status Register enables a core to access the status of the inputs on the distributor.

Usage constraints

A core can only read the status of its own **PPI** and therefore cannot read the status of **PPI** for other cores.

Configurations

Available in all configurations.

Attributes

Base: `0xD00`

Name: PPI Status

Type: -

Reset: `0x00000000`

Width: 32

The following figure shows the PPI Status Register bit assignments.

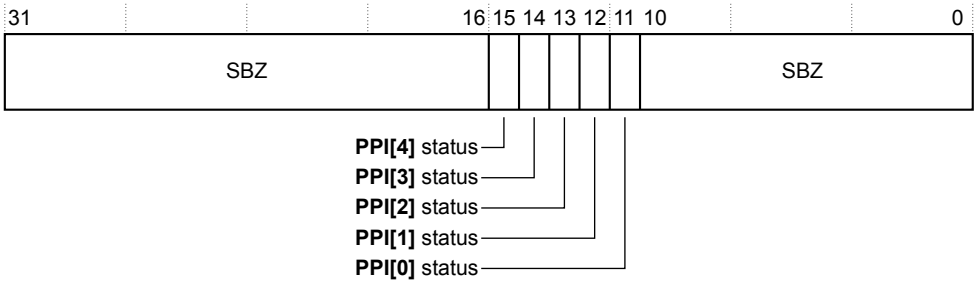


Figure 9-22 PPI Status Register bit assignments

The following table shows the PPI Status Register bit assignments.

Table 9-24 PPI Status Register bit assignments

Bits	Name	Function
[31:16]	-	Reserved. SBZ.
[15:11]	ppi_status	Returns the status of the **PPI[4:0]** inputs on the distributor: **PPI[4]** **nIRQ**. **PPI[3]** Private watchdog. **PPI[2]** Private timer. **PPI[1]** **nFIQ**. **PPI[0]** Global timer. **PPI[1]** and **PPI[4]** are active LOW. **PPI[0]**, **PPI[2]**, and **PPI[3]** are active HIGH. ——— **Note** ——— These bits return the actual status of the **PPI[4:0]** signals. The ICDISPRn and ICDICPRn registers can also provide the **PPI[4:0]** status but because you can write to these registers then they might not contain the actual status of the **PPI[4:0]** signals. ———
[10:0]	-	Reserved. SBZ.

Related reference
Distributor register summary table on page 9-191

SPI Status Registers

The SPI Status Register enable a core to access the status of **IRQS[m:0]** inputs on the distributor, where **m** is an increment of 32.

Usage constraints
There are no usage constraints.

Configurations
Available in all configurations.

Attributes

Base: `0xD04-0xD3C`

Name: SPI Status

Type: RO

Reset: `0x00000000`

Width: 32

The following figure shows the SPI Status Register bit assignments.

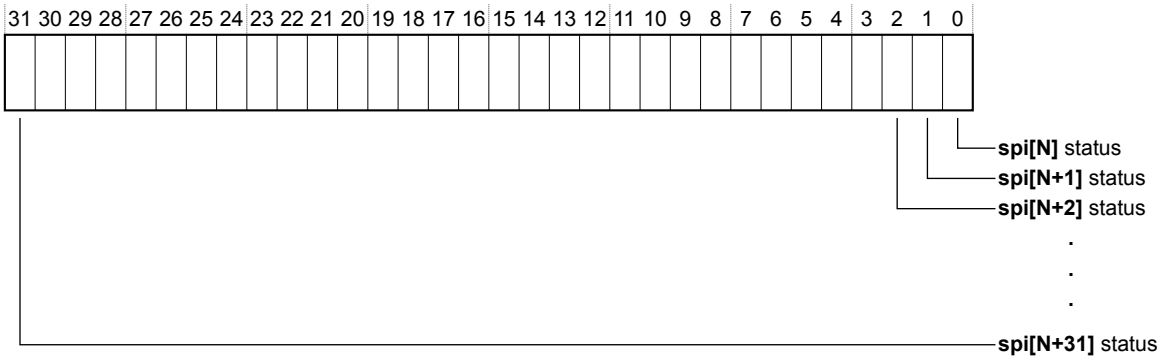


Figure 9-23 SPI Status Register bit assignments

The following table shows the SPI Status Register bit assignments.

Table 9-25 SPI Status Register bit assignments

Bits	Name	Description
[31:0]	spi_status	Returns the status of the **IRQS[m:0]** inputs on the distributor: **Bit[X] = `0b0`** **IRQS[X]** is LOW. **Bit[X] = `0b1`** **IRQS[X]** is HIGH. **Note** The **IRQS** that **X** refers to depends on its bit position and the base address offset of the SPI Status Register as in the following figure shows. These bits return the actual status of the **IRQS** signals. The ICDISPRn and ICDICPRn Registers can also provide the **IRQS** status but because you can write to these registers then they might not contain the actual status of the **IRQS** signals.

The following figure shows the address map that the distributor provides for the SPIs.

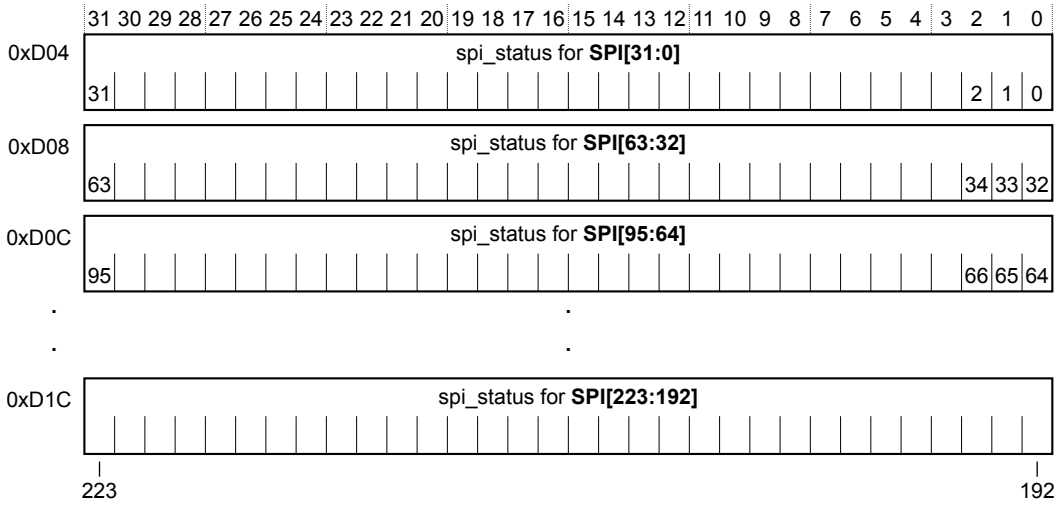


Figure 9-24 SPI Status Register address map

In in this figure the values for the SPIs are read-only. This register contains the values for the SPIs for the corresponding core 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 `0x0` when it reads from these bits.

Related reference

Distributor register summary table on page 9-191

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 Cortex-R8 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.

The following table shows the offset value, register number, and description that are associated with each Peripheral Identification Register.

Table 9-26 Peripheral Identification Registers

Offset	Register number	Value	Description
`0xFD0`	1012	`0x04`	Peripheral Identification Register 4
`0xFD4`	1013	-	Reserved
`0xFD8`	1014	-	Reserved
`0xFDC`	1015	-	Reserved
`0xFE0`	1016	`0x18`	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

The following table shows the offset value, register number, and value that are associated with each Component Identification Register.

Table 9-27 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

Related concepts

2.5.8 Private memory region on page 2-48

9.4.5 Interrupt interface register descriptions

Descriptions of the registers that each Cortex-R8 core interface provides.

Core interface register summary table

Summary of the core interface registers. These registers are word accessible. Any other access is UNPREDICTABLE.

This table does not reproduce information about registers already described in the *Arm® Generic Interrupt Controller Architecture Specification*.

Table 9-28 Core 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[az]
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	*CPU Interface Implementer Identification Register* on page 9-199

CPU Interface Implementer Identification Register

The ICCIIDR Register provides information about the implementer and the revision of the controller.

Usage constraints

There are no usage constraints.

Configurations

Available in all configurations.

Attributes

Base: 0x0FC

Name: ICCIIDR

Type: RO

Reset: 0x3901243B

Width: 32

The following figure shows the ICCIIDR bit assignments.

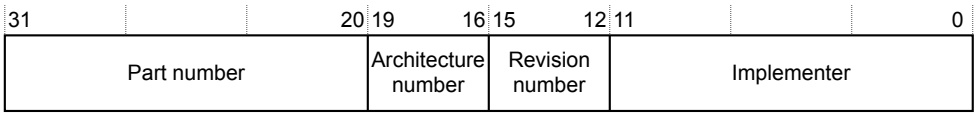


Figure 9-25 ICCIIDR bit assignments

The following table shows the ICCIIDR bit assignments

[az] Only the top four bits of each 8-bit field of the register are in use.

Table 9-29 ICCIIDR bit assignments

Bits	Values	Name	Description
[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	Returns the JEP 106 code of the company that implemented the Cortex-R8 processor interface RTL. It uses the following construct: **[11:8]** JEP 106 continuation code of the implementer. **[7]** 0. **[6:0]** JEP 106 code [6:0] of the implementer.

Related reference

Core interface register summary table on page 9-199

Related concepts

2.5.8 Private memory region on page 2-48

9.5 Private timer and watchdog

The private timer and watchdog blocks have a 32-bit counter that generates an interrupt when it reaches zero, and an eight-bit prescaler value to qualify the clock period.

Other features of the private timer and watchdog blocks include:

- 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, by configuring the **CLK**, **PERIPHCLK**, and **PERIPHCLKEN** signals.

9.5.1 Calculating timer intervals

The timer interval 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.

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.

The following table shows the timer and watchdog registers. All registers not described in the table are Reserved.

Table 9-30 Timer and watchdog registers

Offset	Type	Reset Value	Description
0x00	RW	0x00000000	*Private Timer Load Register* on page 9-202
0x04	RW	0x00000000	*Private Timer Counter Register* on page 9-202
0x08	RW	0x00000000	*Private Timer Control Register* on page 9-202
0x0C	RW	0x00000000	*Private Timer Interrupt Status Register* on page 9-203
0x20	RW	0x00000000	*Watchdog Load Register* on page 9-203
0x24	RW	0x00000000	*Watchdog Counter Register* on page 9-203
0x28	RW	0x00000000	*Watchdog Control Register* on page 9-204
0x2C	RW	0x00000000	*Watchdog Interrupt Status Register* on page 9-205
0x30	RW	0x00000000	*Watchdog Reset Status Register* on page 9-206
0x34	WO	-	*Watchdog Disable Register* on page 9-206

Note

The private timers stop counting when the associated core 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-R8 processor core timer is in debug state, the counter only decrements when the core 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

Private Timer Control Register bit assignments.

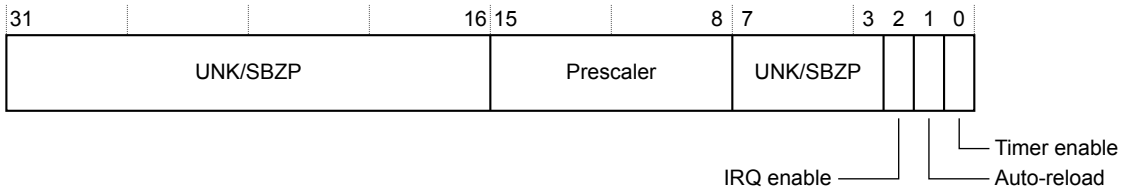


Figure 9-26 Private Timer Control Register bit assignments

The following table shows the Private Timer Control Register bit assignments.

Table 9-31 Private Timer Control Register bit assignments

Bits	Name	Function
[31:16]	-	Reserved. UNK/SBZP.
[15:8]	Prescaler	The prescaler modifies the clock period for the decrementing event for the Counter Register. See *9.5.1 Calculating timer intervals* on page 9-201 for the equation.
[7:3]	-	Reserved. UNK/SBZP.
[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.

Table 9-31 Private Timer Control Register bit assignments (continued)

Bits	Name	Function	
[1]	Auto-reload	Auto-reload enable:	
		`0b0`	Single-shot mode. Counter decrements down to zero, sets the event flag and stops.
		`0b1`	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:	
		`0b0`	Timer is disabled and the counter does not decrement. All registers can still be read and written.
		`0b1`	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. **PERIPHCLK** is the reference clock for this.

Private Timer Interrupt Status Register

Private Timer Interrupt Status Register bit assignments.

This is a banked register for all Cortex-R8 processor cores 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].

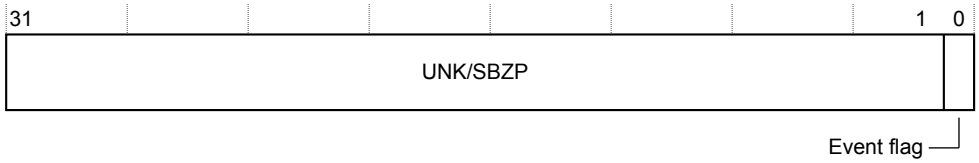


Figure 9-27 Private Timer Interrupt Status Register bit assignments

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-R8 processor core associated with the Watchdog is in debug state, the counter does not decrement until the core 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-R8 processor design.

Watchdog Control Register

Watchdog Control Register bit assignments.

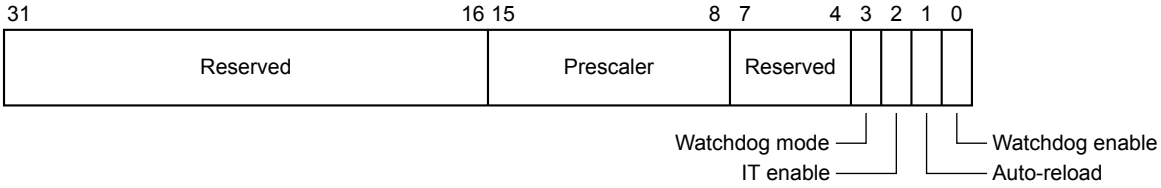


Figure 9-28 Watchdog Control Register bit assignments

The following table shows the Watchdog Control Register bit assignments.

Table 9-32 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 *9.5.1 Calculating timer intervals* on page 9-201.
[7:4]	-	Reserved.
[3]	Watchdog mode	Watchdog mode enable: `0b0` 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 *Watchdog Disable Register* on page 9-206. `0b1` Watchdog mode.
[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.

Table 9-32 Watchdog Control Register bit assignments (continued)

Bits	Name	Function	
[1]	Auto-reload	Auto-reload enable:	
		`0b0`	Single-shot mode. Counter decrements down to zero, sets the event flag and stops.
		`0b1`	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:	
		`0b0`	Watchdog is disabled and the counter does not decrement. All registers can still be read or written.
		`0b1`	Watchdog is enabled and the counter decrements normally.

Watchdog Interrupt Status Register

Watchdog Interrupt Status Register bit assignments.

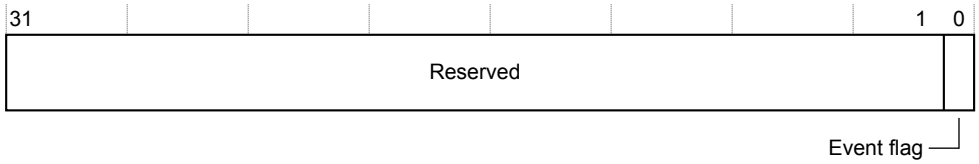


Figure 9-29 Watchdog Interrupt Status Register bit assignments

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.

Watchdog Reset Status Register

Watchdog Reset Status Register bit assignments.

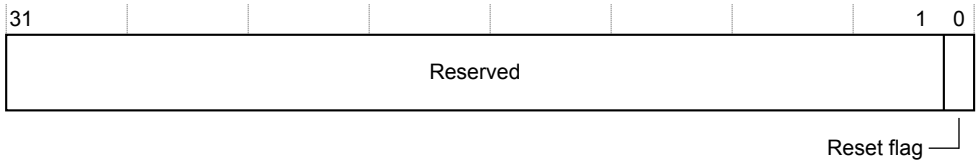


Figure 9-30 Watchdog Reset Status Register bit assignments

In watchdog mode, the reset flag is a sticky bit that is automatically set when the Counter Register reaches zero and a reset request is sent accordingly.

For lock-step or split/lock in lock-step mode, the reset must not be applied immediately to the entire Cortex-R8 processor. This is because after reset the sticky flag is set in one core but not the others and this leads to the assertion of **COMPFAULT**. Therefore, if one core has its watchdog flag set, the other core or cores must reach the same state, that is, also having their watchdog flag set.

The reset flag is cleared when written with a value of one. Trying to write a zero to the reset flag or a one when it is not set has no effect. This flag is not reset by normal Cortex-R8 processor resets but has its own reset line, **nWDRESET**. Do not assert **nWDRESET** when the Cortex-R8 processor reset assertion is the result of a watchdog reset request with **WDRESETREQ**. This distinction enables software to differentiate between a normal boot sequence, reset flag is zero, and one caused by a previous watchdog time-out, reset flag set to one.

Watchdog Disable Register

Use the Watchdog Disable Register to switch from watchdog to timer mode. The software must write `0x12345678` then `0x87654321` successively to the Watchdog Disable Register so that the watchdog mode bit in the Watchdog Control Register is set to zero.

If one of the values written to the Watchdog Disable Register is incorrect or if any other write occurs in between the two word writes, the watchdog remains in its current state. To reactivate the Watchdog, the software must write 1 to the watchdog mode bit of the Watchdog Control Register.

Related reference

Watchdog Control Register on page 9-204

Related concepts

2.5.8 Private memory region on page 2-48

Private Timer Load Register on page 9-202

Private Timer Counter Register on page 9-202

Watchdog Load Register on page 9-203

Related reference

A.3 Reset signals on page Appx-A-360

Private Timer Control Register on page 9-202

Private Timer Interrupt Status Register on page 9-203

Watchdog Counter Register on page 9-203

Watchdog Control Register on page 9-204

Watchdog Interrupt Status Register on page 9-205

Watchdog Reset Status Register on page 9-206

Watchdog Disable Register on page 9-206

Related concepts

2.3 Clocking, resets, and initialization on page 2-29

9.6 Global timer

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, and is accessible to all Cortex-R8 processor cores in the Cortex-R8 processor design. Each core 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-R8 processor cores in a design use the banked ID, ID27, for this interrupt. ID27 is sent to the interrupt controller as a Private Peripheral Interrupt.

Each core can only access its own comparator registers. It cannot access the comparator of another core.

The interrupt from a comparator only goes to the associated core. A core cannot see the interrupt from the comparator of another core.

The global timer is clocked by **PERIPHCLK**.

Note

The global timer does not stop counting when any of the cores are in debug state.

9.6.1 Global timer registers

Summary of the global timer registers. The offset is relative to PERIPH_BASE_ADDR + `0x0200`. Use **nPERIPHRESET** to reset these registers.

Table 9-33 Global timer registers

Offset	Type	Reset value	Description
`0x00`	RW	`0x00000000`	*Global Timer Counter Registers* on page 9-207
`0x04`	RW	`0x00000000`	
`0x08`	RW	`0x00000000`	*Global Timer Control Register* on page 9-208
`0x0C`	RW	`0x00000000`	*Global Timer Interrupt Status Register* on page 9-209
`0x10`	RW	`0x00000000`	*Comparator Value Registers* on page 9-209
`0x14`	RW	`0x00000000`	
`0x18`	RW	`0x00000000`	*Auto-increment Register* on page 9-209

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

Global Timer Control Register bit assignments.

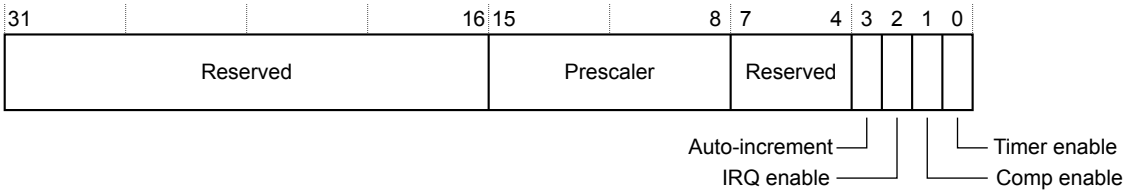


Figure 9-31 Global Timer Control Register bit assignments

The following table shows the Global Timer Control Register bit assignments.

Table 9-34 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 *9.5.1 Calculating timer intervals* on page 9-201 for the equation.
[7:4]	-	Reserved.
[3]	Auto-increment[ba]	This bit is banked per Cortex-R8 processor core: `0b0` Single shot mode. Sets the event flag when the counter is greater than or equal to the comparator value. **Note** It is the responsibility of software to update the comparator value to generate another event based on a higher comparator value. `0b1` 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.
[2]	IRQ enable	This bit is banked per Cortex-R8 processor core. If set, the interrupt ID 27 is set as pending in the Interrupt Distributor when the event flag is set in the Timer Status Register.

[ba] When the Auto-increment and Comp enable bits are set, an IRQ is generated every auto-increment register value.

Table 9-34 Global Timer Control Register bit assignments (continued)

Bits	Name	Function
[1]	Comp enable[ba]	This bit is banked per Cortex-R8 processor core. If set, it allows the comparison between the 64-bit Timer Counter and the related 64-bit Comparator Register.
[0]	Timer enable	Timer enable: `0b0` Timer is disabled and the counter does not increment. All registers can still be read and written. `0b1` Timer is enabled and the counter increments normally.

Global Timer Interrupt Status Register

This is a banked register for all Cortex-R8 processor cores present.

The event flag is a sticky bit that is automatically set when the Counter Register is greater than or equal to 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 with the value 1.

Note

If the Counter Register is greater than or equal to the Comparator Register value when the event flag is cleared, then the event flag is set again after it is cleared. It is the responsibility of software to update the Comparator Register value to a value greater than the current counter value before clearing the event flag.

The following figure shows the Global Timer Interrupt Status Register bit assignments.

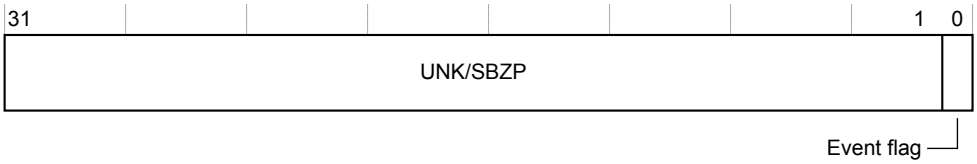


Figure 9-32 Global Timer Interrupt Status Register bit assignments

Comparator Value Registers

There are two 32-bit registers, the lower 32-bit comparator value register at offset **0x10** and the upper 32-bit comparator value register at offset **0x14**.

You must access these registers with 32-bit accesses. You cannot use `STRD`/`LDRD`. There is a Comparator Value Register for each Cortex-R8 processor core.

To ensure that updates to this register do not set the Interrupt Status Register, proceed as follows:
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-R8 processor core 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.

Related concepts

2.5.8 Private memory region on page 2-48

Related reference

9.4.4 Distributor register descriptions on page 9-190

9.7 Accelerator Coherency Port

The *Accelerator Coherency Port* (ACP) reduces software cache maintenance operations when sharing memory regions with other masters, and allows other masters to allocate data into the L2 cache.

This section contains the following subsections:

9.7.1 Coherent and noncoherent mode

The coherency between the ACP traffic and the L1 data cache of the cores is triggered when **AxUSERSC[0]** = 1 and **AxCACHESC[1]** = 1.

- **AxUSERSC[0]** is the shared bit.
- **AxCACHESC[1]** = 1 indicates it is an NC, WT, WB, or a WBWA memory region.
- **AxCACHESC[1]** = 0 indicates it is a SO, or DV memory region. Using the **AxCACHESC[1]** bit prevents SO and DV memory regions from taking part in the coherency mechanism, that is, SCU lookups, on the ACP.

The **x** in the signal name represents either **R** for read or **W** for write.

9.7.2 Read accesses in coherent mode

Behavior of read accesses from the ACP in coherent mode.

- If data is present in the L1 data cache of a core, the data is returned from the L1 data cache of the core that has the data.
- If data is not present in the L1 data cache of a core, the data is returned from the L2 memory.

9.7.3 Write accesses in coherent mode

Behavior of write accesses from the ACP in coherent mode.

- If data is present in the L1 data cache of a core, 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 cleaned and invalidated, and the write is done on the L2 memory.
- If data is not present in the L1 data cache of a core, the data is written to the L2 memory.

9.7.4 AXI protocol configurability, xIDSC, and AxUSERSC

The **xIDSC** bus width value is configurable and must be greater than or equal to 4.

The following table shows the **AWUSERSC[5:0]** encoding for the ACP.

Table 9-35 AWUSERSC encoding for ACP

AWUSERSC value	Information propagated on AXI master port 0, AXI master port 1, and AXI low-latency peripheral port	Usage
[0]	Yes	See *9.7.1 Coherent and noncoherent mode* on page 9-211.
[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.

The following table shows the **ARUSERSC[4:0]** encoding for the ACP.

Table 9-36 ARUSERSC encoding for ACP

ARUSERSC value	Information propagated on AXI master port 0, AXI master port 1, and AXI low-latency peripheral port	Usage
[0]	Yes	See *9.7.1 Coherent and noncoherent mode* on page 9-211
[4:1]	Yes	Ignored by the SCU

Related reference

9.7.1 Coherent and noncoherent mode on page 9-211

9.7.5 AXI protocol restrictions

The ACP is AXI compliant, but not does support the full protocol.

The following AXI protocol restrictions apply:

- 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 nonlocked write. **AWLOCKSC** 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.
- 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, **AxLENSC** = `0x3`, **AxSIZESC** = `0x3`, **AxBURSTSC** = `0b01` (incr), and **AxADDRSC[4:0]** = `0b00000`.
  - For 32 bytes, **AxLENSC** = `0x3`, **AxSIZESC** = `0x3`, **AxBURSTSC** = `0b10` (wrap), and **AxADDRSC[2:0]** = `0b000`.
  - For 16 bytes, **AxLENSC[3:0]** = `0x1`, **AxSIZESC[1:0]** = `0x3`, **AxBURSTSC[1:0]** = `0b01` (incr), and **AxADDRSC[3:0]** = `0b0000`.
  - For 8 bytes: **AxLENSC** = `0x0`, **AxSIZESC** = `0x3`, **AxBURSTSC** = any, and **AxADDRSC** = any.

  The **x** in the signal name 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.

Related concepts

7.4.2 ECC on external AXI bus on page 7-137

Chapter 10
Monitoring, Trace, and Debug

This chapter describes the monitoring, trace, and debug features of the Cortex-R8 processor.

It contains the following sections:

10.1 Performance Monitoring Unit

The Cortex-R8 processor PMU provides eight counters to gather statistics on the operation of the core and memory system. Each counter can count any of the 64 events available in each core.

This section contains the following subsections:

10.1.1 PMU register mappings

The PMU counters, and their associated control registers, are accessible from the internal CP15 interface and from the Debug APB interface.

The following table 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	-	PMXEVTYPER0
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	0x41184000	PMCR

Table 10-1 Performance monitoring instructions and Debug APB mapping (continued)

Debug APB interface mapping	CP15 instruction	Access	Reset	Name
0xE08	0, c9, c14, 0	RW[bb]	0x00000000	PMUSERENR
-	0, c9, c12, 5	RW	-	PMSELR

10.1.2 PMU management registers

The PMU management registers define the standardized set of registers that is implemented by all CoreSight components.

The following table shows the contents of the PMU management registers for the Cortex-R8 processor debug unit.

Table 10-2 PMU management registers

Offset	Register number	Access	Mnemonic	Description
0xD00-0xDFC	832-895	RO	-	*Processor ID Registers* on page 10-216
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	1005	RO	LOCKSTATUS	-
0xFB8	1006	RO	AUTHSTATUS	-
0xFBC-0xFC4	1007-1009	-	-	RAZ
0xFC8	1010	RO	DEVID	Device Identifier
0xFCC	1011	RO	DEVTYPE	-
0xFD0-0xFFC	1012-1023	R	-	*CoreSight™ Identification Registers* on page 10-217

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.

The following table 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	MIDR	RO	0x410FC183	Main ID Register
0xD04	833	CTR	RO	0x8333C003	Cache Type Register
0xD08	834	TCMTR	RO	0x80010001[bc] 0x00000000[bd]	TCM Type Register

[bb] Read only in user mode.
[bc] If TCMs are present.
[bd] If TCMs are not present.

Table 10-3 Processor Identifier Registers (continued)

Offset (hex)	Register number	Mnemonic	Access	Register value	Description
0xD0C	835	MIDR alias	RO	0x410FC183	TLB Type Register
0xD10	836	MPUIR	RO	**12 MPU regions** 0x00000c00 **16 MPU regions** 0x00001000 **20 MPU regions** 0x00001400 **24 MPU regions** 0x00001800	MPU Type Register
0xD14	837	MPIDR	RO	0x80000n0m[be]	Multiprocessor Affinity Register
0xD18	838	REVIDR	UNK	0x00000000	Revision ID Register
0xD1C	839	MIDR alias	RO	0x410FC183	Alias of Main ID Register
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	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

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 Cortex-R8 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.

The following table shows the offset value, register number, value, and description that are associated with each Peripheral Identification Register.

[be] n = **CLUSTERID** input m = core number (0x0 for core 0, 0x1 for core 1, 0x10 for core 2, 0x11 for core 3).

Table 10-4 Peripheral Identification Registers

Offset (hex)	Register number	Value	Description
0xFD0	0x3F4	0x04	Peripheral Identification Register 4
0xFD4	0x3F5	-	Reserved
0xFD8	0x3F6	-	Reserved
0xFDC	0x3F7	-	Reserved
0xFE0	0x3F8	0xB9	Peripheral Identification Register 0
0xFE4	0x3F9	0xB9	Peripheral Identification Register 1
0xFE8	0x3FA	0xrB[bf]	Peripheral Identification Register 2
0xFEC	0x3FB	0x00	Peripheral Identification Register 3

The following table 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	0x3FC	0x0D	Component Identification Register 0
0xFF4	0x3FD	0x90	Component Identification Register 1
0xFF8	0x3FE	0x05	Component Identification Register 2
0xFFC	0x3FF	0xB1	Component Identification Register 3

PMU APB interface

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

[bf] r represents the variant. For r0p1, this is 0.

Table 10-6 PMU register names and APB addresses (continued)

PMU register name	Debug APB address
PMU event type 3	0x40C
PMU event type 4	0x410
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

10.1.3 Performance monitoring events

The Cortex-R8 processor implements the Armv7-A and Armv7-R architectural events. The PMU provides an additional set of Cortex-R8 processor core-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 Cortex-R8 processor, and visible externally through the **PMUEVENTx** bus, where **n** is 0, 1, 2 or 3.

The ETM can export two signals that are connected to the processor PMU. These signals are **ETMEXTOUT[1]** and **ETMEXTOUT[2]** events. See *Chapter 11 Embedded Trace Macrocell on page 11-238* for more information about the **ETMEXTOUT** signals from the ETM.

The following table shows the Cortex-R8 processor events, with their associated event number, and position in the **PMUEVENT** bus.

Table 10-7 Cortex-R8 processor events

Event	Description	Position in PMUEVENT bus
Common events		
0x00	Software increment	[0]
0x01	Instruction cache miss	[1]
0x03	Data cache miss	[2]
0x04	Data cache access	[3]
0x06	Data read	[4]
0x07	Data write	[5]
0x08	Instruction architecturally executed	[11:6]

Table 10-7 Cortex-R8 processor events (continued)

Event	Description	Position in PMUEVENT bus
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	**ETMEXTOUT[1]**	Not applicable
0x41	**ETMEXTOUT[2]**	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]
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]
0x6D	Correctable ECC errors on AXI TCM port	[48]
0x6E	Correctable ECC errors on local AXI fast peripheral port	[49]
Software events		
0x80	STREX passed	[35]

Table 10-7 Cortex-R8 processor events (continued)

Event	Description	Position in PMUEVENT bus
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[bg]	Data cache dependent stall[bh]	[44]
0x97[bg]	Non-Cacheable no peripheral dependent stall[bi]	[45]
0x98[bg]	Non-Cacheable peripheral dependent stall[bj]	[46]
0x99[bg]	Data cache high priority dependent stall[bk]	[47]
0x9A	Accesses to AXI fast peripheral port (reads and writes)	[50]
Reserved		
-	Reserved, tied LOW	[55:51]

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 Arm®v7-A and Arm®v7-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* on page 7-128 for more information.

Related reference

A.9 Performance monitoring signals on page Appx-A-382
Chapter 11 Embedded Trace Macrocell on page 11-238
Chapter 7 Fault Detection on page 7-128

bg 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.
bh The counter counts stalls on AXI M0 with low-priority cacheable traffic.
bi The counter counts stalls on AXI M0 or M1 with low-priority NC/SO/Dev.
bj The counts stall on AXI MP with high-priority SO/DEV.
bk The counts stall on AXI M1 with high-priority when a local SRAM is in use.

10.2 Memory Reconstruction Port

The MRP is an optional feature for the Cortex-R8 processor. All write accesses, regardless of memory attributes, such as Strongly Ordered, Device, Non-Cacheable, and cacheable, are exported from the core 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.

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 core 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 *A.16 Memory reconstruction port signals* on page Appx-A-404 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 core, 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.

Related reference

4.3.10 Auxiliary Control Register on page 4-80
A.16 Memory reconstruction port signals on page Appx-A-404

10.3 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 Cortex-R8 processor in an embedded system.

A single core Cortex-R8 processor has a single ETM. In a multiprocessor configuration, each core can either share a single ETM, or each core can have its own ETM.

The ETM is enabled through the APB debug interface. You can disable the ETM, and power it off for power saving.

See *Chapter 11 Embedded Trace Macrocell* on page 11-238 for more information.

Related reference

Chapter 11 Embedded Trace Macrocell on page 11-238

10.4 Debug

The Cortex-R8 processor implements the Armv7 debug architecture and the set of debug events.

Refer to the *Arm® Architecture Reference Manual Arm®v7-A and Arm®v7-R edition* for the debug architecture and debug events.

Note

When the low-latency interrupt mode is enabled, Arm recommends that you set DBGDSCR.INTdis register bit, to disable interrupts before allowing the processor to enter debug state.

This section contains the following subsections:

10.4.1 Debug events

Depending on the programming of the debug control registers, debug events can generate debug exceptions, that is, software Monitor debug, and make the processor enter debug state, that is, hardware halting debug.

The Cortex-R8 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-R8 processor implements:

- Vector catch.
- BKPT instruction.
- External debug request.
- Halt request.

Note

The Cortex-R8 processor does not implement the OS catch debug event.

See the *Arm® Architecture Reference Manual Arm®v7-A and Arm®v7-R edition* for more information.

10.4.2 Debug registers

Technical reference information for the debug registers.

Register interfaces

The Cortex-R8 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

Mapping for the debug registers.

All other registers are described in the *Arm® Architecture Reference Manual Arm®v7-A and Arm®v7-R edition*.

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[bl]	-[bm]
128	No access	No access	0, c1, c0, 0	RO	DBGDRAR[a]	-
256	No access	No access	0, c2, c0, 0	RO	DBGDSAR[a]	-
1	No access	No access	0, c0, c1, 0	RO	DBGDSCRint[ab]	-
5	No access	No access	0, c0, c5, 0	RO	DBGDTRRXint[a]	-
	No access	No access		WO	DBGDTRTXint[a]	-
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

[bl] Baseline CP14 interface. This register also has an external view through the memory-mapped interface and the CP14 interface.
[bm] Accessible in user mode if bit[12] of the DBGSCR is clear. Also accessible in privileged modes.

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[bn]
576-831	-	-	-	-	Reserved	-
832-895	0xD00-0xDFC	-	-	-	-	*Processor ID Registers* on page 10-216
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	-

[bn] PMU registers are part of the CP15 interface. Reads from the extended CP14 interface return zero. See *4.2 Register summary* on page 4-59. See also *10.1 Performance Monitoring Unit* on page 10-215.

Table 10-8 Debug register mapping (continued)

Register number	APB offset	APB access	CP14 address	CP14 access	Register name	Description
1009	0xFC4	RO	No access	No access	DBGDEVID1	-
1010	0xFC8	RO	No access	No access	DBGDEVID0	-
1011	0xFCC	RO	No access	No access	DBGDEVTYPE	Device Type Register
1012-1016	0xFD0-0xFEC	RO	No access	No access	PERIPHERALID	*CoreSight Identification Registers* on page 10-232
1017-1019	-	-	-	-	Reserved	-
1020-1023	0xFF0-0xFFC	RO	No access	No access	COMPONENTID	*CoreSight Identification Registers* on page 10-232

Debug register descriptions

Register features that are specific to the Cortex-R8 processor.

See the *Arm® Architecture Reference Manual Arm®v7-A and Arm®v7-R edition* for information about other register features not described in this section.

Debug ID Register, DBGDIDR

The DBGDIDR specifies the version of the Debug architecture that is implemented, and some features of the debug implementation.

Usage constraints

There are no usage constraints.

Accessible when the processor is powered down.

For more information about the debug implementation, see *Debug Device ID Register 1* on page 10-229 and *Debug Device ID Register 0* on page 10-230.

Configurations

Available in all configurations.

Attributes

See the register summary in *Debug register mapping table* on page 10-225.

The following figure shows the DBGDIDR bit assignments.

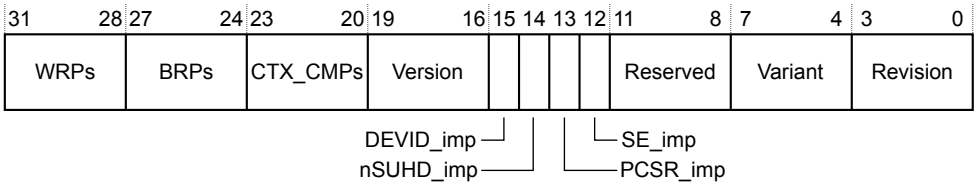


Figure 10-1 DBGDIDR bit assignments

The following table shows the DBGDIDR bit assignments.

Table 10-9 DBGDIDR bit assignments

Bits	Name	Function
[31:28]	WRPs	Indicates the number of *Watchpoint Register Pairs* (WRPs) implemented: `0x3` The processor implements 4 WRPs.
[27:24]	BRPs	Indicates the number of *Breakpoint Register Pairs* (BRPs) implemented: `0x5` The processor implements 6 BRPs.
[23:20]	CTX_CMPs	Indicates the number of BRPs that can be used for Context matching: `0x1` The processor implements 2 breakpoints with Context matching.
[19:16]	Version	Indicates the Debug architecture version: `0x3` The processor implements Armv7 Debug architecture.
[15]	DEVID_imp	Indicates if the Debug Device ID Register (DBGDEVID) is implemented: **1** DBGDEVID is implemented.
[14]	nSUHD_imp	Indicates if the Secure User Halting Debug is implemented: **0** Secure User halting debug is implemented.
[13]	PCSR_imp	Indicates if the Program Counter Sampling Register (DBGPCSR) implemented as register 33: **1** DBGPCSR is implemented as register 33.
[12]	SE_imp	Security Extensions implemented bit: **0** Security Extensions are not implemented.
[11:8]	-	Reserved.
[7:4]	Variant	Indicates the variant number of the processor. This number is incremented on functional changes. The value matches bits [23:20] of the CP15 Main ID Register. For more information, see *4.3.1 Main ID Register* on page 4-70.
[3:0]	Revision	Indicates the revision number of the processor. This number is incremented on bug fixes. The value matches bits [3:0] of the CP15 Main ID Register. For more information, see *4.3.1 Main ID Register* on page 4-70.

Related reference

Debug Device ID Register 1 on page 10-229
Debug Device ID Register 0 on page 10-230
Debug register mapping table on page 10-225
4.3.1 Main ID Register on page 4-70

Debug Status and Control Register, DBGDSCR

Bit exceptions for DBGDSCR.

DBGDSCR behaves as described in the *Arm® Architecture Reference Manual Arm®v7-A and Arm®v7-R edition*, except for the following bits:

PipeAdv, bit[25]

This bit is set each time a branch is resolved in the core.

HALTED, bit[0]

This bit is the only bit of the register that is not reset on debug logic reset. It is reset to `0b0` on a core logic reset. Its behavior is as described in the *Arm® Architecture Reference Manual Arm®v7-A and Arm®v7-R edition*.

Debug Run Control Register, DBGDRCR

Bit exceptions for DBGDRCR.

DBGDRCR behaves as described in the *Arm® Architecture Reference Manual Arm®v7-A and Arm®v7-R edition*, 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 Arm®v7-A and Arm®v7-R edition*.

Device Powerdown and Reset Control Register, DBGPRCR

Bit exceptions for DBGPRCR.

DBGPRCR behaves as described in the *Arm® Architecture Reference Manual Arm®v7-A and Arm®v7-R edition*, except for the following bits:

Bits[2:1]

Not implemented, RAZ/WI.

DBGnoPWRDWN, bit[0]

Implemented as described in the *Arm® Architecture Reference ManualARMv7-A and Arm®v7-R edition* (RW).

Device Powerdown and Reset Status Register, DBGPRSR

Bit exceptions for DBGPRSR.

DBGPRSR behaves as described in the *Arm® Architecture Reference Manual Arm®v7-A and Arm®v7-R edition*, except for the following bits:

Sticky Reset Status, bit[3]

Implemented as described in the *Arm® Architecture Reference Manual Arm®v7-A and Arm®v7-R edition*.

Reset Status, bit[2]

Implemented as described in the *Arm® Architecture Reference Manual Arm®v7-A and Arm®v7-R edition.*

Sticky Powerdown Status, bit[1]

Not implemented, RAZ/WI.

Power-up Status, bit[0]

Implemented, RAO.

Debug Device ID Register 1

The DBGDEVID1 adds to the information given by the DBGDIDR, by describing other features of the debug implementation.

Usage constraints

There are no usage constraints.

Accessible when the processor is powered down.

Attributes

See the register summary in *Debug register mapping table* on page 10-225.

The following figure shows the DBGDEVID1 bit assignments.

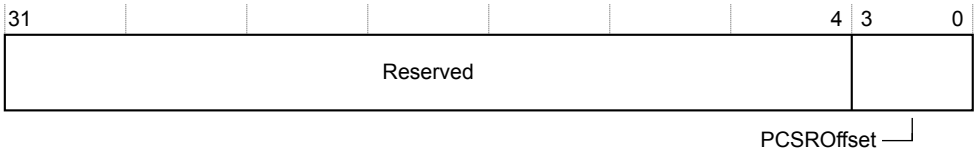


Figure 10-2 DBGDEVID1 bit assignments

The following table shows the DBGDEVID1 bit assignments.

Table 10-10 DBGDEVID1 bit assignments

Bits	Name	Function
[31:4]	-	Reserved.
[3:0]	PCSROffset	Defines the offset applied to DBGPCSR samples: `0b0001` DBGPCSR samples have no offset applied.

Related reference

Debug register mapping table on page 10-225

Debug Device ID Register 0

The DBGDEVID0 extends the DBGDIDR by describing other features of the debug implementation.

Usage constraints

Use in conjunction with DBGDIDR to find the features of the debug implementation. See the Debug ID Register for the DBGDIDR bit assignments.

Accessible when the processor is powered down.

Configurations

Available in all configurations.

Attributes

See the register summary in *Debug register mapping table* on page 10-225.

Figure 9-20 shows the DBGDEVID0 bit assignments.

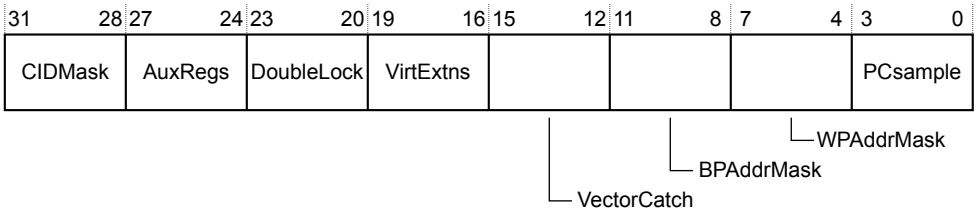


Figure 10-3 DBGDEVID0 bit assignments

Table 9-19 shows the DBGDEVID0 bit assignments.

Table 10-11 DBGDEVID0 bit assignments

Bits	Name	Function
[31:28]	CIDMask	This field indicates the level of support for the Context ID matching breakpoint masking capability. `0b0000` Context ID masking not implemented.
[27:24]	AuxRegs	Specifies support for the Debug External Auxiliary Control Register. `0b0000` The processor does not support the Debug External Auxiliary Control Register.
[23:20]	DoubleLock	Specifies support for the Debug OS Double Lock Register: `0b0000` The Debug OS Double-lock Register is not supported.
[19:16]	VirExtns	Specifies the implementation of the Virtualization Extensions to the Debug architecture: `0b0000` Virtualization Extensions to the Debug architecture are not implemented.
[15:12]	VectorCatch	Defines the form of the vector catch event implemented: `0b0000` The processor implements address matching form of vector catch.
[11:8]	BPAddrMask	Indicates the level of support for the *Immediate Virtual Address* (IVA) matching breakpoint masking capability: `0b1111` Breakpoint address masking not implemented. DBGBCRn[28:24] are UNK/SBZP.
[7:4]	WPAddrMask	Indicates the level of support for the DVA matching watchpoint masking capability: `0b0001` Watchpoint address mask implemented.
[3:0]	PCSample	Indicates the level of support for Program Counter sampling using debug registers 40 and 41: `0b0000` Program Counter Sampling Register (DBGPCSR) is not implemented as register 40, and Context ID Sampling Register (DBGCIDSR) is not implemented.

Related reference

Debug register mapping table on page 10-225
Debug ID Register, DBGDIDR on page 10-227

Breakpoint and Watchpoint Registers, DBGBVRn, DBGBCRn, DBGWVRn, and DBGWCRn

Breakpoint and Watchpoint Registers features that are specific to the Cortex-R8 processor.

The Breakpoint and Watchpoint Registers behave as described in the *Arm® Architecture Reference Manual Arm®v7-A and Arm®v7-R edition*, 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 is the debug logic reset signal. This signal must be asserted during a power up 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.

The following table shows the contents of the debug management registers for the Cortex-R8 processor debug unit. On the Cortex-R8 processor, the debug management registers are memory-mapped.

Table 10-12 Debug management registers

APB offset	Register number	Access	Mnemonic	Description
0xD00-0xDFC	832-895	RO	-	*Processor ID Registers* on page 10-232
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	-
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	-	*CoreSight Identification Registers* on page 10-232

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.

The *Processor Identifier Registers* table shows the APB offset value, register number, mnemonic, and description that are associated with each Processor ID Register.

Related reference

Processor ID Registers on page 10-216

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 Cortex-R8 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.

The following table shows the APB offset value, register number, and description that are associated with each Peripheral Identification Register.

Table 10-13 Peripheral Identification Registers for core 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	0x18	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

The following table shows the APB offset value, register number, and value that are associated with each Component Identification Register.

Table 10-14 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

10.4.3 External debug interface

External debug interface signals.

The following figure shows the external debug interface signals.

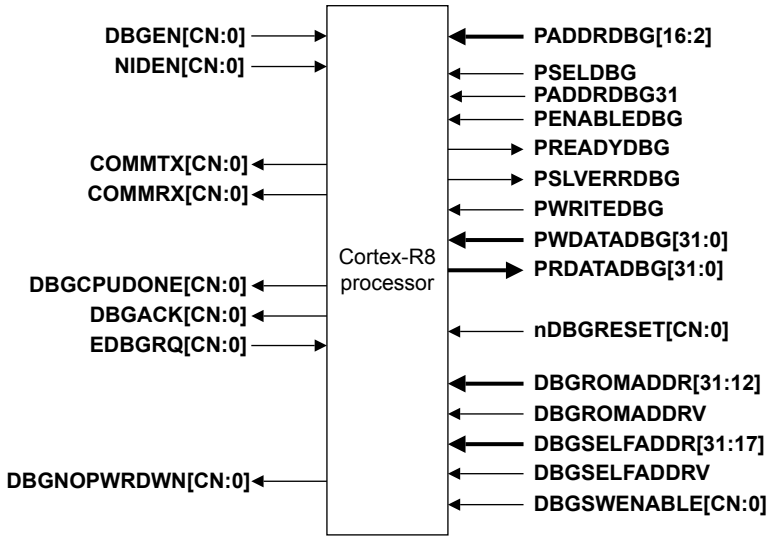


Figure 10-4 External debug interface signals

Authentication signals

Table showing the valid combinations of authentication signals along with their associated debug permissions.

Table 10-15 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-R8 processor APB slave port. This APB slave interface supports 32-bit wide data, stalls, and slave-generated aborts.

The following table shows the mapping of **PADDRDBG[16:2]**.

Table 10-16 PADDRDBG[16:2] mapping

Address map	Reset domain	Component
`0x00000-0x00FFF`	Integration	ROM Table.
`0x01000-0x0FFFF`	-	Reserved.
`0x10000-0x10FFF`	CPU0	Core 0 debug. See *Debug register mapping table* on page 10-225 for debug resource memory mapping.
`0x11000-0x11FFF`	CPU0	Core 0 PMU. See *10.1 Performance Monitoring Unit* on page 10-215 for PMU resource mapping.
`0x12000-0x12FFF`	CPU1	Core 1 Debug. See *Debug register mapping table* on page 10-225 for debug resource memory mapping.
`0x13000-0x13FFF`	CPU1	Core 1 PMU. See *10.1 Performance Monitoring Unit* on page 10-215 for PMU resource mapping.
`0x14000-0x14FFF`	CPU2	Core 2 Debug. See *Debug register mapping table* on page 10-225 for debug resource memory mapping.
`0x15000-0x15FFF`	CPU2	Core 2 PMU. See *10.1 Performance Monitoring Unit* on page 10-215 for PMU resource mapping.
`0x16000-0x16FFF`	CPU3	Core 3 Debug. See *Debug register mapping table* on page 10-225 for debug resource memory mapping.
`0x17000-0x17FFF`	CPU3	Core 3 PMU. See *10.1 Performance Monitoring Unit* on page 10-215 for PMU resource mapping.
`0x18000-0x18FFF`	Integration	Core 0 CTI.
`0x19000-0x19FFF`	Integration	Core 1 CTI.
`0x1A000-0x1AFFF`	Integration	Core 2 CTI.
`0x1B000-0x1BFFF`	Integration	Core 3 CTI.
`0x20000-0x1BFFF`	-	Reserved.
`0x1C000-0x1CFFF`	Integration	Core 0 ETM.

Table 10-16 PADDRDBG[16:2] mapping (continued)

Address map	Reset domain	Component
0x1D000-0x1DFFF	Integration	Core 1 ETM.
0x1E000-0x1EFFF	Integration	Core 2 ETM.
0x1F000-0x1FFFF	Integration	Core 3 ETM.

The **PADDRDBG31** signal indicates the source of the access to the core.

External debug request interface

The external debug request interface signals assert various debug states and indicate debug events.

EDBGRQ

This signal generates a halting debug event by requesting the core 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 core.

DBGACK

The core 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 core has completed a *Data Synchronization Barrier* (DSB) as part of the entry procedure to debug state.

If the debugger writes 1 to the **DBGDSCR.DBGack** bit when the **DBGDSCR.HALTED** bit equals 1, the processor asserts **DBGCPUDONE** after it has completed all non-debug state memory accesses. The system uses **DBGCPUDONE** as an indicator that all memory accesses issued by the core are a result of operations performed by a debugger.

The following figure shows the Cortex-R8 processor connections specific to debug request and restart and the CoreSight inputs and outputs.

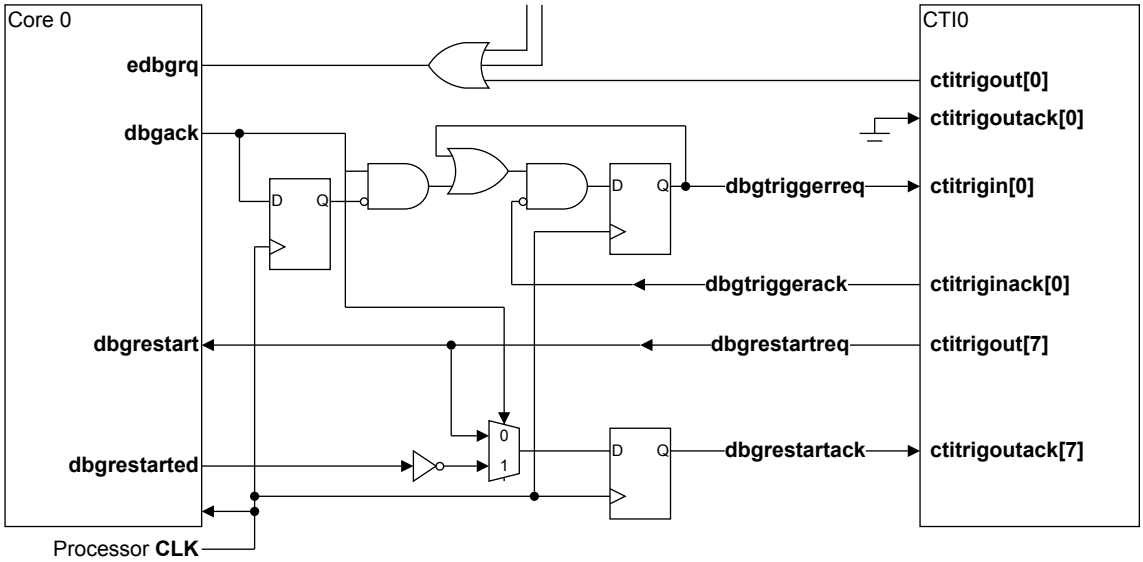


Figure 10-5 Debug request restart-specific connections

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 core to read, and is deasserted when the core 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 core writes the data. Its value is equal to the inverse of the DBGDSCR[29] DTRTX full flag.

DBGROMADDR and DBGSELFADDR

Cortex-R8 cores 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 core itself.

Related reference

A.14 External debug signals on page Appx-A-397

10.4.4 Trigger inputs and outputs

Trigger inputs and outputs that are available to the CTI.

The following table shows the CTI inputs.

Table 10-17 Trigger inputs

CTI input	Name	Description
0	**DBGTRIGGER**, pulsed	Pulsed on entry to debug state
1	**PMUIRQ**	PMU generated interrupt
2	**ETMEXTOUT[0]**	ETM external output

Table 10-17 Trigger inputs (continued)

CTI input	Name	Description
3	**ETMEXTOUT[1]**	ETM external output
4	**ETMEXTOUT[2]**	ETM external output
5	**ETMEXTOUT[3]**	ETM external output
6	**COMMTX**	Debug communication transmit channel is empty
7	**COMMRX**	Debug communication receive channel is full

The following table shows the CTI outputs.

Table 10-18 Trigger outputs

CTI output	Name	Description
0	**EDBGRQ**	Causes the core to enter debug state
1	**ETMEXTIN[0]**	ETM external input - ETM event
2	**ETMEXTIN[1]**	ETM external input - ETM event
3	**ETMEXTIN[2]**	ETM external input - ETM event
4	**ETMEXTIN[3]**	ETM external input - ETM event
5	-	-
6	**nCTIIRQ**	CTI interrupt

Chapter 11
Embedded Trace Macrocell

This chapter describes the Embedded Trace Macrocell for the Cortex-R8 processor.

It contains the following sections:

11.1 About the ETM

The ETM provides real-time instruction trace and data trace for the Cortex-R8 processor. The Cortex-R8 processor ETM generates information that trace software tools use to reconstruct the execution of all or part of a program.

For full reconstruction of program execution, the Cortex-R8 processor ETM is able to trace:

- All instructions, including condition code pass/fail.
- Load/store address and data values.
- Values of context-ID.
- Target addresses of taken direct and indirect branch operations.
- Exceptions.
- Changes in core instruction set state.
- Entry to and return from Debug state when Halting Debug-mode is enabled.
- Cycle counts relating to instruction execution.

The Cortex-R8 processor ETM contains logic, known as resources, that enable you to control tracing by specifying the exact set of triggering and filtering conditions required for a particular application. Resources include address comparators and data value comparators, counters, and a sequencer.

The Cortex-R8 processor ETM is a CoreSight component. For more information about CoreSight, see the *Arm® Embedded Trace Macrocell Architecture Specification ETMv4*.

This section contains the following subsections:

11.1.1 The CoreSight™ debug environment

The Cortex-R8 processor ETM is designed for use with CoreSight, an extensible, system-wide debug and trace architecture from Arm.

See the *Arm® CoreSight™ SoC-400 User Guide* for more information about how to use the Cortex-R8 processor ETM in a full CoreSight system.

A software debugger provides the user interface to the Cortex-R8 processor ETM. You can use this interface to:

- Configure Cortex-R8 processor ETM facilities such as filtering.
- Configure optional trace features such as cycle accurate tracing.
- Configure the other CoreSight components such as the *Trace Port Interface Unit* (TPIU).
- Access the core debug and performance monitor units.

A CoreSight system can provide memory-mapped access from the core to its own debug and trace components.

The Cortex-R8 processor ETM outputs its trace stream to the AMBA 3 *Advanced Trace Bus* (ATB) interfaces. The CoreSight infrastructure provides the following options:

- Export the trace information through a trace port. An external *Trace Port Analyzer* (TPA) captures the trace information as the figure that follows shows.
- Write the trace information directly to an on-chip *Embedded Trace Buffer* (ETB) or to system memory. You can read out the trace at low speed using a JTAG or Serial Wire interface when the trace capture is complete as the figure shows.

The debugger extracts the executed image from memory and the captured trace information from the TPA or ETB and decompresses the image to provide full disassembly, with symbols, of the code that was executed. The trace information generated by the Cortex-R8 processor ETM gives the debugger the

capability to link this data back to the original high-level source code, to provide a visualization of how the code was executed on the Cortex-R8 processor.

The figure shows how the Cortex-R8 processor ETM fits into a CoreSight debug environment to provide full trace capabilities in a single core system. In this example, the external debug software configures the trace and debug components through the *Debug Access Port* (DAP). The ROM table contains a unique identification code for the SoC and the base addresses of the components connected to the debug APB. The trace stream from the Cortex-R8 processor ETM is replicated to provide on-chip storage using the CoreSight ETB or output off-chip using the TPIU. Cross-triggering operates through the *Cross Trigger Interface* (CTI) and *Cross Trigger Matrix* (CTM) components.

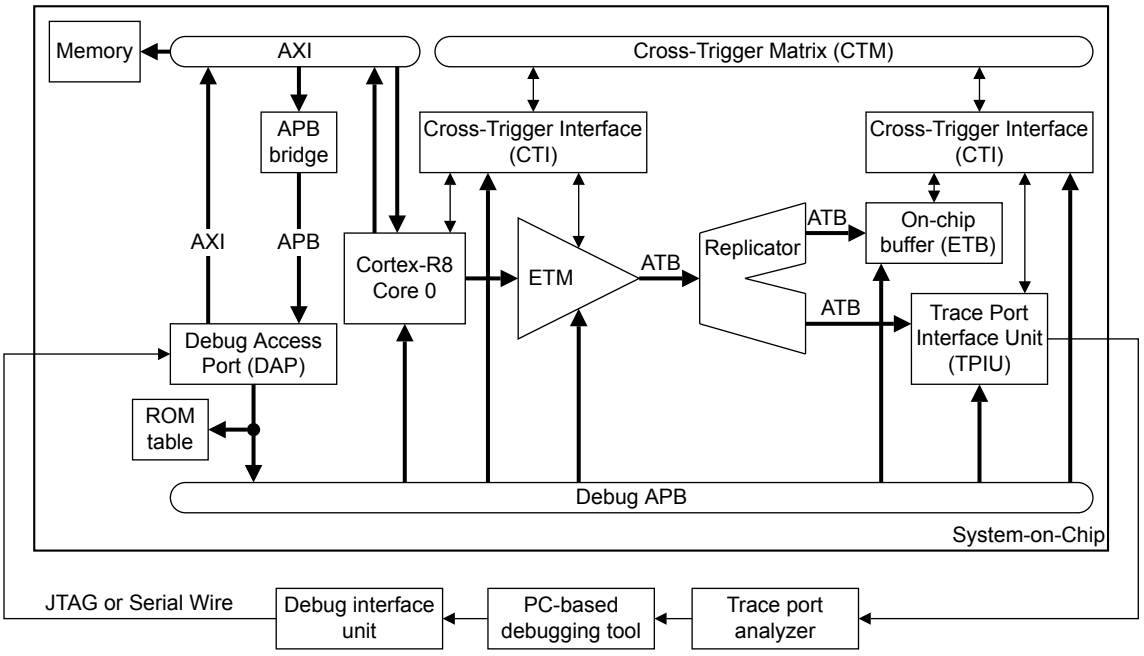


Figure 11-1 ETM system diagram for a single core Cortex-R8 processor

Note

In the figure, the arrows on the thick lines show the transaction direction on buses, from master to slave port. Each bus contains individual signals that go from master to slave and other signals that go from slave to master.

For a Cortex-R8 processor implementation with several cores, the following options are available:

- One ETM, statically shared between each core.
- One ETM dedicated to each core.

11.1.2 Features

The Cortex-R8 processor ETM supports tracing of 32-bit Arm instructions, and 16-bit and 32-bit Thumb instructions.

See the *Arm® Embedded Trace Macrocell Architecture Specification ETMv4* for information about:

- The trace protocol.
- The features of ETMv4.
- Controlling tracing using triggering and filtering resources.
- ETM sharing.

The following table shows the features of the Cortex-R8 processor ETM that are implementation-defined, in terms of either:

- The number of times the feature is implemented.
- The size of the feature.

Table 11-1 Cortex-R8 processor ETM features with implementation-defined number of instances or size

Feature	Cortex-R8 processor ETM value	Notes
Address comparators	4 pairs	See bits[3:0] of the *11.8.34 ID Register 4* on page 11-304
Data value comparators	2	See bits[7:4] of the *11.8.34 ID Register 4* on page 11-304
Context ID comparators	1	See bits[27:24] of the *11.8.34 ID Register 4* on page 11-304
Single-Shot comparator resource	2, one for instruction, one for data	See bits[2:0] of the *11.8.38 Single-Shot Comparator Status Registers 0-1* on page 11-308
Counters	2	See bits[30:28] of the *11.8.35 ID Register 5* on page 11-305
Cycle count size	12	See bits[28:25] of the *11.8.32 ID Register 2* on page 11-301
Sequencer	1	One four-state sequencer. See bits[27:25] of the *11.8.35 ID Register 5* on page 11-305.
Core comparator inputs	Not implemented	See bits[15:12] of the *11.8.34 ID Register 4* on page 11-304
External inputs	64	See bits[8:0] of the *11.8.35 ID Register 5* on page 11-305
External outputs	4	See bits[3:0] of the *11.8.7 Event Control 1 Register* on page 11-274
External input selectors	4	See bits[11:9] of the *11.8.35 ID Register 5* on page 11-305
Resource selector pairs	8	See bits[19:16] of the *11.8.34 ID Register 4* on page 11-304
Instruction trace port size	32-bit	-
Data trace port size	64-bit	-
Instruction FIFO[bo]	128 byte with 32-bit output	Uses ATB
Data FIFO	256 byte with 64-bit output	Uses ATB
Claim tag bits	4	See bits[3:0] of the *11.8.50 Claim Tag Set Register* on page 11-320

The following table shows the optional features of the ETM architecture that the Cortex-R8 processor ETM implements.

Table 11-2 Cortex-R8 processor ETM implementation of optional features

Feature	Implemented?	Notes
Configurable FIFO	No	-
Trace Start/Stop block	Yes	*11.8.16 ViewInst Start/Stop Control Register* on page 11-283
Trace all branches option	Yes	See bit[5] of the *11.8.30 ID Register 0* on page 11-298
Trace of conditional instructions	Yes	See bits[13:12] and bit[6] of the *11.8.30 ID Register 0* on page 11-298, using the full CPSR value
Cycle counting in instruction trace	Yes	See bit[7] of the *11.8.30 ID Register 0* on page 11-298
Data trace supported	Yes	See bits[4:3] of the *11.8.30 ID Register 0* on page 11-298

[bo] Instruction trace can be configured to take priority over data trace. See bit[10] of the TRCSTALLCTLR.

Table 11-2 Cortex-R8 processor ETM implementation of optional features (continued)

Feature	Implemented?	Notes
Data address comparison	Yes	See bit[8] of the *11.8.34 ID Register 4* on page 11-304
OS Lock mechanism	Yes	*11.8.39 OS Lock Access Register* on page 11-310
Secure non-invasive debug	No	The Cortex-R8 processor does not implement the Security Extensions
Context ID tracing	Yes	See bits[9:5] of the *11.8.32 ID Register 2* on page 11-301
Trace output	Yes	ATB
Timestamp size (48/64)	System configurable	See bits[28:24] of the *11.8.30 ID Register 0* on page 11-298
Memory mapped access to ETM registers	Yes	-
System instruction access to ETM registers	No	-
VMID comparator support	No	See bits[31:28] of the *11.8.34 ID Register 4* on page 11-304
ATB trigger support	Yes	See bit[22] of the *11.8.35 ID Register 5* on page 11-305

Related reference
11.8.34 ID Register 4 on page 11-304
11.8.38 Single-Shot Comparator Status Registers 0-1 on page 11-308
11.8.35 ID Register 5 on page 11-305
11.8.32 ID Register 2 on page 11-301
11.8.7 Event Control 1 Register on page 11-274
11.8.50 Claim Tag Set Register on page 11-320
11.8.16 ViewInst Start/Stop Control Register on page 11-283
11.8.30 ID Register 0 on page 11-298
11.8.39 OS Lock Access Register on page 11-310
Appendix A Signal Descriptions on page Appx-A-356

11.1.3 Interfaces

The Cortex-R8 processor ETM has main interfaces for processor trace, ATB, Debug APB, and test.

Related reference
11.3 Interfaces on page 11-246

11.1.4 Configurable options

The Cortex-R8 processor ETM provides **NUMPROC**, **SYSSTALL**, and **TSSIZE** configuration input signals.

See the *Arm® Embedded Trace Macrocell Architecture Specification ETMv4* for more information.

11.1.5 Test features

The Cortex-R8 processor ETM provides the **DFTSE** input for testing the implemented device.

See the *Arm® Cortex®-R8 MPCore Processor Integration Manual*.

Integration test registers are provided for testing the Cortex-R8 processor ETM integration in a SoC and performing CoreSight topology detection.

Related reference

11.8.61 Integration Test Registers on page 11-332

11.2 Functional description

Functional description of the Cortex-R8 processor ETM.

The following figure shows the main functional blocks of the Cortex-R8 processor ETM.

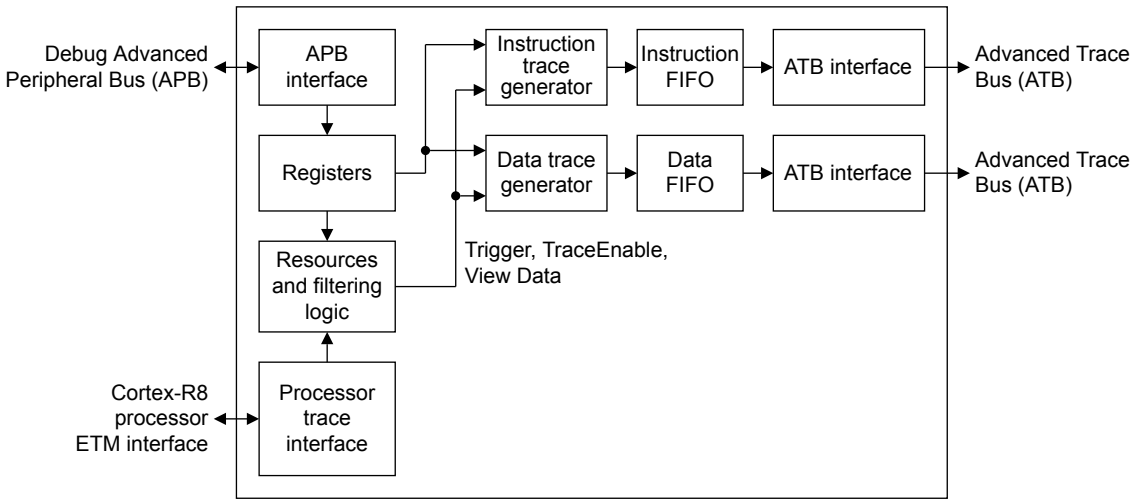


Figure 11-2 Cortex-R8 processor ETM block diagram

11.2.1 Processor interface

The processor interface block connects to the Cortex-R8 processor ETM interface. It tracks the execution and speculation information from the core, decodes the control signals and passes on the information to the internal interfaces.

11.2.2 Instruction trace generator

The instruction trace generator block generates the trace packets that are a compressed form of the execution information provided by the Cortex-R8 processor trace generation. The trace packets are then passed to the FIFO.

11.2.3 Data trace generator

The data trace generator block generates trace packets that are a compressed form of the external data transfer provided by the Cortex-R8 processor trace generation. The trace packets are then passed to the FIFO.

11.2.4 FIFO

The FIFO block buffers bursts of trace packets. Separate FIFOs are provided for instruction and data trace streams.

11.2.5 Resources and filtering logic

The resources and filtering logic blocks contain various comparators and state machines that are programmed by trace software to trigger and filter the trace information. They start and stop trace generation, depending on the conditions that have been set.

11.2.6 ATB interface

The ATB interface block reads up to four or eight bytes of packet information from the FIFO and sends them over the ATB interface.

11.2.7 APB interface

The APB interface block implements the interface to the APB, that provides access to the programmable registers. It provides address decoding and pipelining of the address and data to and from the APB.

11.2.8 Global timestamping

The Cortex-R8 processor ETM supports connection to a global timestamp source. This provides a 48-bit or 64-bit timestamp which can be used for coarse-grained profiling, and correlation of trace sources. Arm recommends that the timestamp counter is no slower than 10% of the Cortex-R8 processor clock.

Note

Decompression of data trace relies on the presence of a global timestamp count.

11.3 Interfaces

A Cortex-R8 processor ETM has two ATB interfaces, an APB interface, a processor trace interface, and a test interface. There are also a number of other miscellaneous interface signals that configure ETM, trace, and debug options.

ATB

Two ATB interfaces, one 32 bits, and one 64 bits wide, used for trace output from the macrocell. These interfaces have handshaking signals that indicate when trace data is valid and when the receiving component is ready to accept data. There are also signals to request and acknowledge a flush of the trace information and to indicate when a trigger condition has occurred.

See the *Arm® AMBA® 4 ATB Protocol Specification ATBv1.0 and ATBv1.1* for more information about these interfaces.

APB

An APB interface that provides access to the programmable registers in the ETM and connects to the system Debug APB. This interface is used to configure the ETM for a trace session.

See the *Arm® AMBA® 4 ATB Protocol Specification ATBv1.0 and ATBv1.1* for more information about this interface.

Processor trace

The Cortex-R8 processor passes its execution information to the ETM over the processor trace interface. This interface provides both instruction and data execution history and contains address, data, and control information. The information carried on the control bus includes:

- The number of instructions executed in the same cycle.
- Changes in program flow.
- The current core instruction state.
- The addresses of memory locations accessed by load and store instructions.
- The data values transferred by load and store instructions.
- The type, direction, and size of a transfer.
- Condition code information.
- Exception information.
- Current context ID.

There is also a context ID bus that indicates the current context ID value of the core.

This interface also includes:

- The **ETMEVENT** bus.
- Wait for interrupt state information signals.
- A signal from the ETM to power up the interface.

Miscellaneous

The ETM has other interface signals that:

- Configure the ETM.
- Input and output external resource information that controls triggering and filtering of the trace stream.
- Control which core is enabled, as the trace source, on the processor trace interface of the ETM.
- Enable invasive and non-invasive debug.

Test

This interface contains the scan enable signal used in production testing of the ETM.

This section contains the following subsection:

- *11.3.1 Core PMU connectivity* on page 11-247.

11.3.1 Core PMU connectivity

Connections of the **ETMEVENT** inputs for each core. These come from the CTI, the Cortex-R8 processor core *Performance Monitoring Unit* (PMU), and ECC monitoring logic, if present.

Table 11-3 ETMEVENT connections

Bits	Description
[63:62]	Not used
[61]	ECC fatal error has occurred in the core RAM
60	ECC fatal error has occurred on one of the Cortex-R8 processor buses: M0, M1, MP, ACP, AXI TCM or FPP0-3, if present
[59:4]	Core performance monitor events
[3:0]	**CTITRIGOUT**

Note

When a single ETM is shared between several cores, the PMU event pins are driven from the relevant core, but the CTI connections for the ETM are always to CTI0, regardless of which core is being traced.

The ETM output resources, **ETMEXTOUT**, are connected to the CTI and also as inputs to the core PMU, as the following table shows.

Table 11-4 ETM EXTOUT connections to CTI and core PMU

ETM output	CTI input	PMU input
ETMEXTOUT[0]	CTITRIGIN**[2]**	-
ETMEXTOUT[1]	**CTITRIGIN[3]**	**PMUEXTIN[0]**
ETMEXTOUT[2]	**CTITRIGIN[4]**	**PMUEXTIN[1]**
ETMEXTOUT[3]	**CTITRIGIN[5]**	-

Related reference

Chapter 10 Monitoring, Trace, and Debug on page 10-214

Related reference

11.3.1 Core PMU connectivity on page 11-247

11.1.4 Configurable options on page 11-242

11.4 Clocking and resets

Reference information for the Cortex-R8 processor ETM clocks and resets.

11.4.1 Cortex®-R8 processor ETM clock

The Cortex-R8 processor ETM has one clock, **CLK**. This clock is synchronous to the **CLK** input of the Cortex-R8 processor.

11.4.2 Cortex®-R8 processor ETM low-power control

The ETM can be configured to remain active even if the Cortex-R8 processor enters a low-power state. See bit[23] of the ID Register 5.

Related reference

11.8.35 ID Register 5 on page 11-305

11.4.3 Cortex®-R8 processor ETM reset

The Cortex-R8 processor ETM has a single reset, **nRESET**, and must only be reset by a debug reset event.

Note

The programming state must be reconfigured after a reset.

11.4.4 Access permissions and power domains

To determine the access permissions as described in the ETM Architecture v4, the ETM implements a single power domain. The ETM (debug) power domain is typically separate from the Cortex-R8 processor core power domain.

11.5 Operation

Description of the implementation-defined features of the Cortex-R8 processor ETM macrocell.

See the *Arm® Embedded Trace Macrocell Architecture Specification ETMv4* for more information about the operation of the Cortex-R8 processor ETM.

This section contains the following subsections:

11.5.1 Implementation-defined registers

There are two groups of ETM registers. Registers that are completely defined by the Arm ETMv4 architecture specification, and registers that are at least partly implementation-defined.

Related reference

11.7 ETM registers on page 11-254

11.5.2 Precise TraceEnable events

The Arm ETMv4 architecture specification states that **ViewInst** and **ViewData** are imprecise under certain conditions, with some implementation-defined exceptions. The only condition which ensures that **ViewInst** and **ViewData** are precise is that the enabling event condition is TRUE.

11.5.3 Parallel instruction execution

The Cortex-R8 processor supports parallel instruction execution. This means that the macrocell is capable of tracing two instructions per cycle.

Although the **ViewInst** is evaluated for each instruction as required, the macrocell does not trace one instruction without the other. In other words, if one instruction is specified to be traced, the instruction it is paired with is always traced as well. If **ViewData** is active, any data associated with the paired instruction is also traced. If **ViewData** selects only one transfer of a multiple load or store, both transfers which are issued by the core as a 64-bit transfer are traced.

11.5.4 Context ID tracing

The Cortex-R8 processor ETM detects updates to the Context ID register and, when the context change sequence is completed with an ISB or exception, traces the appropriate number of bytes as a context ID packet as part of the instruction trace stream. Coprocessor register transfers never generate data trace.

11.5.5 Trace and comparator features

In ETM Architecture v4, it is IMPLEMENTATION DEFINED whether an ETM supports several trace and comparator features.

Trace features

The Cortex-R8 processor ETM implements all the ETMv4 trace features.

This means it supports:

- Data value and data address tracing.
- Data suppression.
- Cycle-accurate tracing.
- Timestamping.

See the *Arm® Embedded Trace Macrocell Architecture Specification ETMv4* for descriptions of these features.

Comparator features

The Cortex-R8 processor ETM implements data address comparison.

See the *Arm® Embedded Trace Macrocell Architecture Specification ETMv4* for a description of data address comparison.

11.5.6 Data address range filtering

When data address range filtering is used to include both loads and stores, or no data address include ranges are selected, an address range is set to exclude loads or stores, but not both. When a `SWP` matches for both the include and exclude ranges, both the load and store transfers are traced.

11.5.7 Interaction with the PMU

The Cortex-R8 processor core includes a *Performance Monitoring Unit* (PMU) that enables events, such as cache misses and instructions executed, to be counted over a period. The macrocell can still use these events by means of the extended external input facility. Each bit in the **ETMEVENT[63:0]** input is mapped to the corresponding extended external input.

Bits[3:0] of **ETMEVENT** are reserved for cross trigger connections. PMU event signals use bits[59:4], ECC signals use bits[61:60] if implemented, and the remainder are free for system- specific use if implemented. Any four of the external inputs can be selected for further use in the resource logic.

PMU event number 8, Executed instruction count, is presented by the core as a 6-bit vector which the ETM is not able to count. These are OR-gated together for connection to the ETM and the ETM is not able to count instruction execution directly.

The Cortex-R8 processor core PMU can count two of the ETM external outputs as additional events. These events are not provided back to the macrocell as extended external inputs.

These facilities enable additional filtering of the system events using ETM resources, such as instruction address ranges or the start/stop resource, before they are passed back to the PMU for counting. To do this:

- Configure the ETM external input selectors to the system events you want to count.
- Configure the required ETM filtering resource as appropriate.
- Configure the ETM external outputs and the required ETM filtering resource.
- Select the ETM external outputs as the events to be counted in the Cortex-R8 processor core PMU.

Related reference

Chapter 10 Monitoring, Trace, and Debug on page 10-214

11.5.8 Packet formats

See the Arm ETMv4 architecture specification for descriptions of the trace packet formats generated by the Cortex-R8 processor ETM.

11.5.9 Resource selection

The Cortex-R8 processor ETM uses event selectors to control trace events (triggers and markers in the trace stream), timestamp events, ViewInst event, ViewData events, Counter controls, and sequencer state transitions.

Each event selector is configured to be sensitive to a resource selector pair, and one resource selector pair can be used to control more than one event selector.

The ETM provides one fixed resource selector pair, with static values of 0 and 1, and seven configurable selector pairs. A resource selector pair provides a bit field OR selector for resources in two different groups, with each group and a configurable boolean combination provided.

The following table shows the resources which can be selected.

Table 11-5 Resource selection

Group	Select	Resource
0b0000	0-3	External input selector 0-3
0b0010	0-1	Counter at zero 0-1
	4-7	Sequencer states 0-3
0b0011	0-1	Single-Shot comparator 0-1
0b0100	0-7	Single address comparator 0-7
0b0101	0-3	Address range comparator 0-3
0b0110	0	Context ID comparator 0

As an example, the following figure shows the steps necessary to use a single address comparator to generate a trigger event and an ATB trigger. This example uses the first single resource selector which can be user-configured.

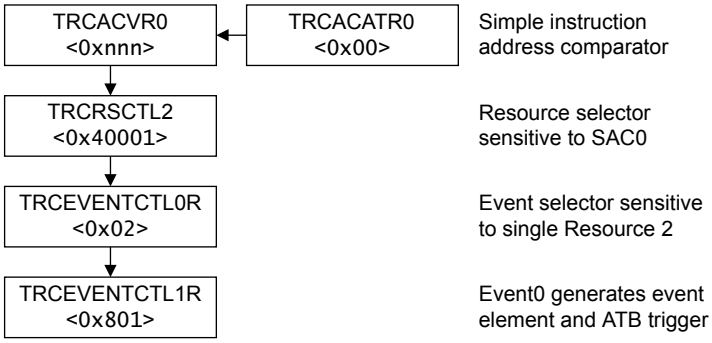


Figure 11-3 Trigger event resource selection

11.5.10 Trace flush behavior

Events which have been observed by the ETM can be confirmed to have reached the trace bus output with the use of the ATB flush protocol. Both ATB ports must be flushed if all trace is required. The ETM internally flushes instruction and data trace together whenever either flush request is seen but does not guarantee that the trace data has drained from the ETM. When the Cortex-R8 processor enters a low-power state, this also causes all trace to be output from the ETM.

If the Cortex-R8 processor enters a low-power state while an ATB flush request is in progress, the flush is acknowledged only after all instructions presented to the ETM have been traced.

11.5.11 Low-power state behavior

When the Cortex-R8 processor enters a low-power state, the ETM resources become inactive after a delay which allows the last instruction executed to trigger a comparator, update the counter or sequencer, and then cause an event packet to be inserted in the trace stream. This event packet is presented on the trace bus before the ETM itself enters a low-power state. If an event packet is generated for a different reason, it is not guaranteed to be output before the ETM enters a low-power state, but is traced when the Cortex-R8 processor leaves the low-power state, if the ETM logic is not reset before this can occur.

This low-power behavior can be disabled, in which case the ETM resources remain active.

11.5.12 Cycle counter

The cycle counter is a 12-bit counter. It does not count when non-invasive debug is disabled, or when the Cortex-R8 processor is in a low-power state.

11.5.13 Micro-architectural exceptions

These exception encodings are intended to permit trace decompression rather than to trace the underlying cause of instruction replay.

The ETM indicates exceptions for the following micro-architectural behaviors using the following encodings:

`0b10000`
Execution replay resulting from incorrect decode prediction.

`0b10001`
Execution replay resulting from ECC error detection.

11.5.14 Synchronization

All sources of synchronization are combined before being used to generate synchronization in both trace streams, if data trace is active.

Periodic synchronization of the data trace stream is aligned with synchronization in the instruction stream. If the ETM is configured to trace only events in the data stream, it is also necessary to configure the instruction trace stream to contain sufficient elements to permit the required data trace stream synchronization.

11.6 Controlling ETM programming

When programming the ETM registers, you must enable all the changes at the same time. For example, if the counter is reprogrammed, it might start to count based on incorrect events, before the trigger condition has been correctly set up.

You program and read the ETM registers using the Debug APB interface. This provides a direct method of programming the Cortex-R8 processor ETM.

You must use the ETM main enable in the TRCPRGCTLR to disable all trace operations during programming. in the following figure shows the procedure to follow.

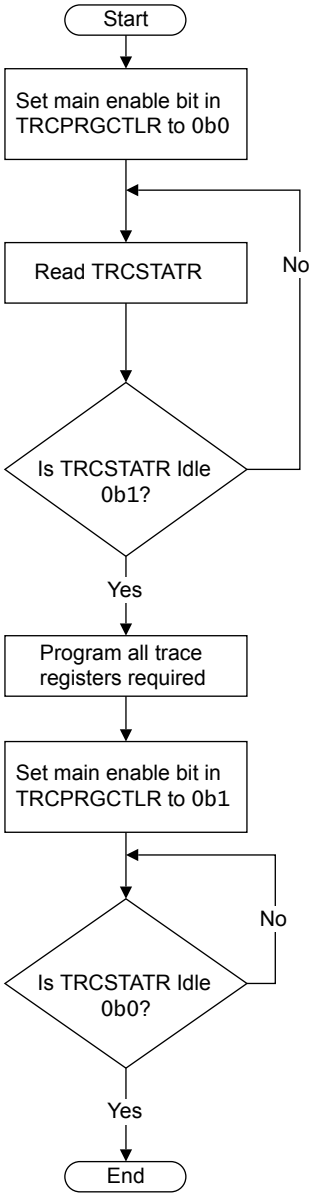


Figure 11-4 Programming Cortex-R8 processor ETM registers

The Cortex-R8 processor does not have to be in the debug state while you program the ETM registers.

Related reference

11.8.1 Programming Control Register on page 11-267

11.7 ETM registers

Cortex-R8 processor ETM register summary, and a list of the macrocell registers organized by functional group.

See the *Arm® Embedded Trace Macrocell Architecture Specification ETMv4* for registers that are not described here.

This section contains the following subsections:

11.7.1 Cortex®-R8 processor ETM register summary

Summary of the ETM registers. The registers are listed in numerical order. Registers not listed here are not implemented.

Note

- Reading a non-implemented register address returns zero. Writing to a non-implemented register address has no effect.
- The Reset value column shows the value of the register immediately after an ETM reset. For read-only registers, every read of the register returns this value.
- Access type is described as follows:

  RW
  Read and write.

  RO
  Read only.

  WO
  Write only.

All ETM registers are 32 bits wide.

Table 11-6 Cortex-R8 processor ETM register summary

Register number	Base offset	Name	Type	Reset value	Description
1	0x004	TRCPRGCTLR	RW	0x00000000	*11.8.1 Programming Control Register* on page 11-267
2	0x008	TRCPROCSELR	RW	0x00000000	*11.8.2 Processor Select Control Register* on page 11-268
3	0x00C	TRCSTATR	RO	-	*11.8.3 Status Register* on page 11-268
4	0x010	TRCCONFIGR	RW	-	*11.8.4 Trace Configuration Register* on page 11-269
6	0x018	TRCAUXCTLR	RW	0x00000000	*11.8.5 Auxiliary Control Register* on page 11-271
8	0x020	TRCEVENTCTL0R	RW	-	*11.8.6 Event Control 0 Register* on page 11-272
9	0x024	TRCEVENTCTL1R	RW	-	*11.8.7 Event Control 1 Register* on page 11-274
11	0x02C	TRCSTALLCTLR	RW	-	*11.8.8 Stall Control Register* on page 11-275
12	0x030	TRCTSCTLR	RW	-	*11.8.9 Global Timestamp Control Register* on page 11-276
13	0x034	TRCSYNCPR	RW	-	*11.8.10 Synchronization Period Register* on page 11-277

Table 11-6 Cortex-R8 processor ETM register summary (continued)

Register number	Base offset	Name	Type	Reset value	Description
14	0x038	TRCCCCTLR	RW	-	*11.8.11 Cycle Count Control Register* on page 11-278
15	0x03C	TRCBBCTLR	RW	-	*11.8.12 Branch Broadcast Control Register* on page 11-279
16	0x040	TRCTRACEIDR	RW	-	*11.8.13 Trace ID Register* on page 11-280
32	0x080	TRCVICTLR	RW	-	*11.8.14 ViewInst Main Control Register* on page 11-281
33	0x084	TRCVIIECTLR	RW	-	*11.8.15 ViewInst Include/Exclude Control Register* on page 11-282
34	0x088	TRCVISSCTLR	RW	-	*11.8.16 ViewInst Start/Stop Control Register* on page 11-283
40	0x0A0	TRCVDCTLR	RW	-	*11.8.17 ViewData Main Control Register* on page 11-284
41	0x0A4	TRCVDSACCTLR	RW	-	*11.8.18 ViewData Include/Exclude Single Address Comparator Register* on page 11-285
42	0x0A8	TRCVDARCCTLR	RW	-	*11.8.19 ViewData Include/Exclude Address Range Comparator Register* on page 11-286
64-66	0x100-0x108	TRCSEQEVRn	RW	-	*11.8.20 Sequencer State Transition Control Registers 0-2* on page 11-287
70	0x118	TRCSEQRSTEVR	RW	-	*11.8.21 Sequencer Reset Control Register* on page 11-288
71	0x11C	TRCSEQSTR	RW	-	*11.8.22 Sequencer State Register* on page 11-289
72	0x120	TRCEXTINSELR	RW	-	*11.8.23 External Input Select Register* on page 11-290
80-81	0x140-0x144	TRCCNTRLDVRn	RW	-	*11.8.24 Counter Reload Value Registers 0-1* on page 11-291
84	0x150	TRCCNTCTLR0	RW	-	*11.8.25 Counter Control Register 0* on page 11-292
85	0x154	TRCCNTCTLR1	RW	-	*11.8.26 Counter Control Register 1* on page 11-293
88-89	0x160-0x164	TRCCNTVRn	RW	-	*11.8.27 Counter Value Registers 0-1* on page 11-294
96	0x180	TRCIDR8	RO	0x00000040	*11.8.28 ID Register 8-13* on page 11-295
97	0x184	TRCIDR9	RO	0x00000040	
98	0x188	TRCIDR10	RO	0x00000040	
99	0x18C	TRCIDR11	RO	0x00000011	
100	0x190	TRCIDR12	RO	0x00000020	
101	0x194	TRCIDR13	RO	0x00000000	
112	0x1C0	TRCIMSPEC0	RW	0x00000000	*11.8.29 Implementation Specific Register 0* on page 11-297
120	0x1E0	TRCIDR0	RO	0xXX001EFF	*11.8.30 ID Register 0* on page 11-298
121	0x1E4	TRCIDR1	RO	0x4100F400	*11.8.31 ID Register 1* on page 11-300
122	0x1E8	TRCIDR2	RO	0x00420084	*11.8.32 ID Register 2* on page 11-301

Table 11-6 Cortex-R8 processor ETM register summary (continued)

Register number	Base offset	Name	Type	Reset value	Description
123	0x1EC	TRCIDR3	RO	0xXX090004	*11.8.33 ID Register 3* on page 11-302
124	0x1F0	TRCIDR4	RO	0x01270124	*11.8.34 ID Register 4* on page 11-304
125	0x1F4	TRCIDR5	RO	0x28C70840	*11.8.35 ID Register 5* on page 11-305
130-140	0x208-0x240	TRCRSCTLRn	RW	-	*11.8.36 Resource Selection Registers 2-16* on page 11-306
160-161	0x280-0x284	TRCSSCCRn	RW	-	*11.8.37 Single-Shot Comparator Control Registers 0-1* on page 11-307
168-169	0x2A0-0x2A4	TRCSSCSRn	RW	-	*11.8.38 Single-Shot Comparator Status Registers 0-1* on page 11-308
192	0x300	TRCOSLAR	WO	-	*11.8.39 OS Lock Access Register* on page 11-310
193	0x304	TRCOSLSR	RO	-	*11.8.40 OS Lock Status Register* on page 11-311
196	0x310	TRCPDCR	RW	0x00000000	*11.8.41 Power Down Control Register* on page 11-312
197	0x314	TRCPDSR	RO	0x00000023	*11.8.42 Power Down Status Register* on page 11-312
256-271	0x400-0x43C	TRCACVRn	RW	-	*11.8.43 Address Comparator Value Registers 0-7* on page 11-313
288-303	0x480-0x4BC	TRCACATRn	RW	-	*11.8.44 Address Comparator Access Type Registers 0-7* on page 11-314
320-321	0x500-0x504	TRCDVCVRn	RW	-	*11.8.45 Data Value Comparator Value Registers 0-1* on page 11-316
352-359	0x580-0x59C	TRCDVCMRn	RW	-	*11.8.46 Data Value Comparator Mask Registers 0-1* on page 11-317
384	0x600	TRCCIDCVR0	RW	-	*11.8.48 Context ID Comparator Value Register 0* on page 11-318
416	0x680	TRCCIDCCTLR0	RW	-	*11.8.47 Context ID Comparator Control Register 0* on page 11-318
951	0xEDC	TRCITMISCOUTR	RW	-	*Integration Miscellaneous Outputs Register* on page 11-334
952	0xEE0	TRCITMISCINR	RO	-	*Integration Miscellaneous Inputs Register* on page 11-335
953	0xEE4	TRCITATBIDR	RW	-	*Integration ATB Identification Register* on page 11-336
954	0xEE8	TRCIRDDATAR	RW	-	*Integration Data ATB Data Register* on page 11-337
955	0xEEC	TRCITIDATAR	RW	-	*Integration Instruction ATB Data Register* on page 11-338
956	0xEF0	TRCITDATBINR	RO	-	*Integration Data ATB In Register* on page 11-339
957	0xEF4	TRCITIATBINR	RO	-	*Integration Instruction ATB In Register* on page 11-340
958	0xEF8	TRCITDATBOUTR	RW	-	*Integration Data ATB Out Register* on page 11-341
959	0xEFC	TRCITIATBOUTR	RW	-	*Integration Instruction ATB Out Register* on page 11-342

Table 11-6 Cortex-R8 processor ETM register summary (continued)

Register number	Base offset	Name	Type	Reset value	Description
960	0xF00	TRCITCTRL	RW	0x00000000	*11.8.49 Integration Mode Control Register* on page 11-319
1000	0xFA0	TRCCLAIMSET	RW	0x00000000	*11.8.50 Claim Tag Set Register* on page 11-320
1001	0xFA4	TRCCLAIMCLR	RW	0x00000000	*11.8.51 Claim Tag Clear Register* on page 11-321
1002	0xFA8	TRCDEVAFF0	RO	-	*11.8.52 Device Affinity Register* on page 11-321
1004	0xFB0	TRCLAR	WO	-	*11.8.53 Software Lock Access Register* on page 11-322
1005	0xFB4	TRCLSR	RO	-	*11.8.54 Software Lock Status Register* on page 11-323
1006	0xFB8	TRCAUTHSTATUS	RO	-	*11.8.55 Authentication Status Register* on page 11-324
1007	0xFBC	TRCDEVARCH	RO	0x47704A17	*11.8.56 Device Architecture Register* on page 11-325
1010	0xFC8	TRCDEVID	RO	0x00000000	*11.8.57 Device ID Register* on page 11-326
1011	0xFCC	TRCDEVTYPE	RO	0x00000013	*11.8.58 Device Type Register* on page 11-327
1012-1019	0xFD0-0xFEC	TRCPIDRn	RO	-	*11.8.59 Peripheral Identification Registers* on page 11-328
1020-1023	0xFF0-0xFFC	TRCCIDRn	RO	-	*11.8.60 Component Identification Registers* on page 11-331

11.7.2 Functional grouping of registers

Summary of the macrocell registers arranged by functional group. These functional groups include all the registers.

The functional group register tables include additional information about each register, including:

- The register access type. This is read-only, write-only, or read and write.
- The base offset address of the register. The base address of a register is always four times its register number.
- Additional information about the implementation of the register, where appropriate.

General control and ID registers

The general control and ID registers in numerical order.

Table 11-7 General control and ID registers

Register number	Name	Base offset	Description
1	TRCPRGCTLR	0x004	*11.8.1 Programming Control Register* on page 11-267
2	TRCPROCSELR	0x008	*11.8.2 Processor Select Control Register* on page 11-268
3	TRCSTATR	0x00C	*11.8.3 Status Register* on page 11-268
4	TRCCONFIGR	0x010	*11.8.4 Trace Configuration Register* on page 11-269
6	TRCAUXCTLR	0x018	*11.8.5 Auxiliary Control Register* on page 11-271
8	TRCEVENTCTL0R	0x020	*11.8.6 Event Control 0 Register* on page 11-272
9	TRCEVENTCTL1R	0x024	*11.8.7 Event Control 1 Register* on page 11-274
11	TRCSTALLCTLR	0x02C	*11.8.8 Stall Control Register* on page 11-275
12	TRCTSCTLR	0x030	*11.8.9 Global Timestamp Control Register* on page 11-276
13	TRCSYNCPR	0x034	*11.8.10 Synchronization Period Register* on page 11-277
14	TRCCCCTLR	0x038	*11.8.11 Cycle Count Control Register* on page 11-278
15	TRCBBCTLR	0x03C	*11.8.12 Branch Broadcast Control Register* on page 11-279
16	TRCTRACEIDR	0x040	*11.8.13 Trace ID Register* on page 11-280

Trace filtering control registers

The trace filtering control registers in numerical order.

Table 11-8 Trace filtering control registers

Register number	Name	Base offset	Description
32	TRCVICTLR	0x080	*11.8.14 ViewInst Main Control Register* on page 11-281
33	TRCVIIECTLR	0x084	*11.8.15 ViewInst Include/Exclude Control Register* on page 11-282
34	TRCVISSCTLR	0x088	*11.8.16 ViewInst Start/Stop Control Register* on page 11-283
40	TRCVDCTLR	0x0A0	*11.8.17 ViewData Main Control Register* on page 11-284
41	TRCVDSACCTLR	0x0A4	*11.8.18 ViewData Include/Exclude Single Address Comparator Register* on page 11-285
42	TRCVDARCCTLR	0x0A8	*11.8.19 ViewData Include/Exclude Address Range Comparator Register* on page 11-286

Derived resource registers

The derived resource registers in numerical order.

These registers control:

- The two counters, and associated events.
- The sequencer, and associated state change events.
- Trigger events.
- EXTOUT (External Output) events.
- Extended External Input selection.

Table 11-9 Derived resource registers

Register number	Name	Base offset	Description
64-66	TRCSEQVRn	`0x100-0x108`	*11.8.20 Sequencer State Transition Control Registers 0-2* on page 11-287
70	TRCSEQRSTEVR	`0x118`	*11.8.21 Sequencer Reset Control Register* on page 11-288
71	TRCSEQSTR	`0x11C`	*11.8.22 Sequencer State Register* on page 11-289
72	TRCEXTINSELR	`0x120`	*11.8.23 External Input Select Register* on page 11-290
80-81	TRCCNTRLDVRn	`0x140-0x144`	*11.8.24 Counter Reload Value Registers 0-1* on page 11-291
84	TRCCNTCTLR0	`0x150`	*11.8.25 Counter Control Register 0* on page 11-292
85	TRCCNTCTLR1	`0x154`	*11.8.26 Counter Control Register 1* on page 11-293
88-89	TRCCNTVRn	`0x160-0x164`	*11.8.27 Counter Value Registers 0-1* on page 11-294

Implementation-specific and identification registers

The implementation-specific and identification registers in numerical order.

Table 11-10 Implementation-specific and identification registers

Register number	Name	Base offset	Description
96	TRCIDR8	0x180	*11.8.28 ID Register 8-13* on page 11-295
97	TRCIDR9	0x184	
98	TRCIDR10	0x188	
99	TRCIDR11	0x18C	
100	TRCIDR12	0x190	
101	TRCIDR13	0x194	
112	TRCIMSPEC0	0x1C0	*11.8.29 Implementation Specific Register 0* on page 11-297
120	TRCIDR0	0x1E0	*11.8.30 ID Register 0* on page 11-298
121	TRCIDR1	0x1E4	*11.8.31 ID Register 1* on page 11-300
122	TRCIDR2	0x1E8	*11.8.32 ID Register 2* on page 11-301
123	TRCIDR3	0x1EC	*11.8.33 ID Register 3* on page 11-302
124	TRCIDR4	0x1F0	*11.8.34 ID Register 4* on page 11-304
125	TRCIDR5	0x1F4	*11.8.35 ID Register 5* on page 11-305

Resource selection registers

The resource selection registers in numerical order.

Table 11-11 Resource selection registers

Register number	Name	Base offset	Description
130-140	TRCRSCTLRn	0x208-0x240	*11.8.36 Resource Selection Registers 2-16* on page 11-306

Single-shot comparator registers

The Single-shot comparator registers in numerical order.

Table 11-12 Single-shot comparator registers

Register number	Name	Base offset	Description
160-161	TRCSSCCRn	0x280-0x284	*11.8.37 Single-Shot Comparator Control Registers 0-1* on page 11-307
168-169	TRCSSCSRn	0x2A0-0x2A4	*11.8.38 Single-Shot Comparator Status Registers 0-1* on page 11-308

OS lock and power control registers

The OS lock and power control registers in numerical order.

Table 11-13 OS lock and power control registers

Register number	Name	Base offset	Description
192	TRCOSLAR	0x300	*11.8.39 OS Lock Access Register* on page 11-310
193	TRCOSLSR	0x304	*11.8.40 OS Lock Status Register* on page 11-311
196	TRCPDCR	0x310	*11.8.41 Power Down Control Register* on page 11-312
197	TRCPDSR	0x314	*11.8.42 Power Down Status Register* on page 11-312

Comparator registers

The comparator registers in numerical order.

Table 11-14 Comparator registers

Register number	Name	Base offset	Description
256-271	TRCACVRn	0x400-0x43C	*11.8.43 Address Comparator Value Registers 0-7* on page 11-313
288-303	TRCACATRn	0x480-0x4BC	*11.8.44 Address Comparator Access Type Registers 0-7* on page 11-314
320-321	TRCDVCVRn	0x500-0x504	*11.8.45 Data Value Comparator Value Registers 0-1* on page 11-316
352-359	TRCDVCMRn	0x580-0x59C	*11.8.46 Data Value Comparator Mask Registers 0-1* on page 11-317
384	TRCCIDCVR0	0x600	*11.8.48 Context ID Comparator Value Register 0* on page 11-318

Integration test registers

The Integration test registers in numerical order.

Table 11-15 Integration test registers

Register number	Name	Base offset	Description
951	Integration Miscellaneous Outputs Register	0xEDC	*Integration Miscellaneous Outputs Register* on page 11-334
952	Integration Miscellaneous Inputs Register	0xEE0	*Integration Miscellaneous Inputs Register* on page 11-335
953	Integration ATB Identification Register	0xEE4	*Integration ATB Identification Register* on page 11-336
954	Integration Data ATB Data Register	0xEE8	*Integration Data ATB Data Register* on page 11-337
955	Integration Instruction ATB Data Register	0xEEC	*Integration Instruction ATB Data Register* on page 11-338
956	Integration Data ATB In Register	0xEF0	*Integration Data ATB In Register* on page 11-339
957	Integration Instruction ATB In Register	0xEF4	*Integration Instruction ATB In Register* on page 11-340
958	Integration Data ATB Out Register	0xEF8	*Integration Data ATB Out Register* on page 11-341
959	Integration Instruction ATB Out Register	0xEFC	*Integration Instruction ATB Out Register* on page 11-342

CoreSight™ management registers

The CoreSight management registers in numerical order.

Table 11-16 CoreSight management registers

Register number	Name	Base offset	Description
960	TRCITCTRL	0xF00	*11.8.49 Integration Mode Control Register* on page 11-319
1000	TRCCLAIMSET	0xFA0	*11.8.50 Claim Tag Set Register* on page 11-320
1001	TRCCLAIMCLR	0xFA4	*11.8.51 Claim Tag Clear Register* on page 11-321
1002	TRCDEVAFF0	0xFA8	*11.8.52 Device Affinity Register* on page 11-321
1004	TRCLAR	0xFB0	*11.8.53 Software Lock Access Register* on page 11-322
1005	TRCLSR	0xFB4	*11.8.54 Software Lock Status Register* on page 11-323
1006	TRCAUTHSTATUS	0xFB8	*11.8.55 Authentication Status Register* on page 11-324
1007	TRCDEVARCH	0xFBC	*11.8.56 Device Architecture Register* on page 11-325
1010	TRCDEVID	0xFC8	*11.8.57 Device ID Register* on page 11-326
1011	TRCDEVTYPE	0xFCC	*11.8.58 Device Type Register* on page 11-327
1012-1019	TRCPIDRn	0xFD0-0xFEC	*11.8.59 Peripheral Identification Registers* on page 11-328
1020-1023	TRCCIDRn	0xFF0-0xFFC	*11.8.60 Component Identification Registers* on page 11-331

11.8 Register descriptions

Reference information for each of the Cortex-R8 processor ETM registers.

This section contains the following subsections:

11.8.1 Programming Control Register

The TRCPRGCTLR enables the Cortex-R8 processor ETM.

Usage constraints

See *11.6 Controlling ETM programming* on page 11-253.

Configurations

Available in all configurations.

Attributes

Register number: 1

Base offset `0x004`

Name: TRCPRGCTLR

Type: RW

Reset: `0x00000000`

The following figure shows the TRCPRGCTLR bit assignments.

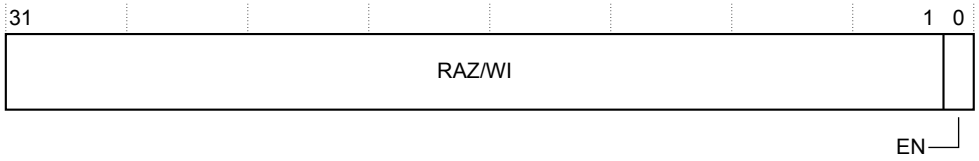


Figure 11-5 TRCPRGCTLR bit assignments

The following table shows the TRCPRGCTLR bit assignments.

Table 11-17 TRCPRGCTLR bit assignments

Bits	Name	Function
[31:1]	-	Reserved. RAZ/WI.
[0]	EN	Trace program enable: `0b0` The external pin **ETMIFENx** is LOW, and clocks are only enabled when necessary to process APB accesses, or drain any already generated trace. Writes to most registers are ignored. This is the reset value. `0b1` The external pin **ETMIFENx** is HIGH, and clocks are enabled except for when the **CPUACTIVE** input is deasserted and all trace has been drained.

Related reference

11.6 Controlling ETM programming on page 11-253

11.7.1 Cortex®-R8 processor ETM register summary on page 11-254

General control and ID registers on page 11-258

11.8.2 Processor Select Control Register

The TRCPROCSELR controls a multiplexer to select trace information from the connected Cortex-R8 processor cores.

Usage constraints

See the *Arm® Embedded Trace Macrocell Architecture Specification ETMv4*.

Configurations

Available in all configurations.

Attributes

Register number: 2

Base offset `0x008`

Name: TRCPROCSELR

Type: RW

Reset: `0x00000000`

The following figure shows the TRCPROCSELR bit assignments.

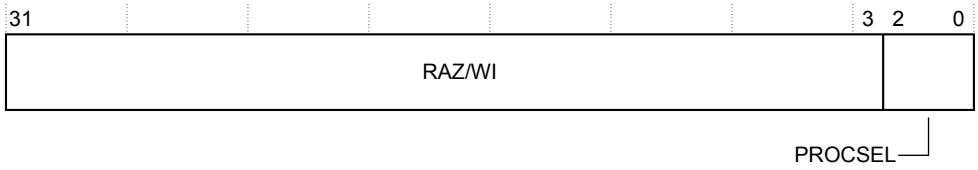


Figure 11-6 TRCPROCSELR bit assignments

The following table shows the TRCPROCSELR bit assignments.

Table 11-18 TRCPROCSELR bit assignments

Bits	Name	Function
[31:3]	-	Reserved. RAZ/WI
[2:0]	PROCSEL	Drives the **PROCSEL[2:0]** pin

Related reference

11.7.1 Cortex®-R8 processor ETM register summary on page 11-254

General control and ID registers on page 11-258

11.8.3 Status Register

The TRCSTATR indicates the Cortex-R8 processor ETM status.

Usage constraints

There are no usage constraints.

Configurations

Available in all configurations.

Attributes

Register number: 3

Base offset `0x00C`

Name: TRCSTATR

Type: RO

Reset: -

The following figure shows the TRCSTATR bit assignments.

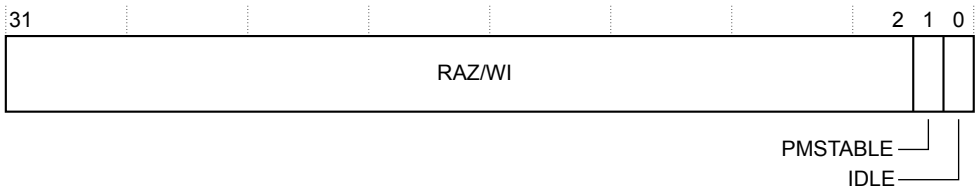


Figure 11-7 TRCSTATR bit assignments

The following table shows the TRCSTATR bit assignments.

Table 11-19 TRCSTATR bit assignments

Bits	Name	Function
[31:2]	-	Reserved. RAZ/WI.
[1]	PMSTABLE	Indicates whether the Cortex-R8 processor ETM registers are stable and can be read: `0b0` The programmers model is not stable. `0b1` The programmers model is stable.
[0]	IDLE	Idle status: `0b0` The Cortex-R8 processor ETM is not idle. `0b1` The Cortex-R8 processor ETM is idle.

Related reference

11.7.1 Cortex®-R8 processor ETM register summary on page 11-254

General control and ID registers on page 11-258

11.8.4 Trace Configuration Register

The TRCCONFIGR sets the basic tracing options for the trace unit.

Usage constraints

There are no usage constraints.

Configurations

Available in all configurations.

Attributes

Register number: 4

Base offset `0x010`

Name: TRCCONFIGR

Type: RW

Reset: -

The following figure shows the TRCCONFIGR bit assignments.

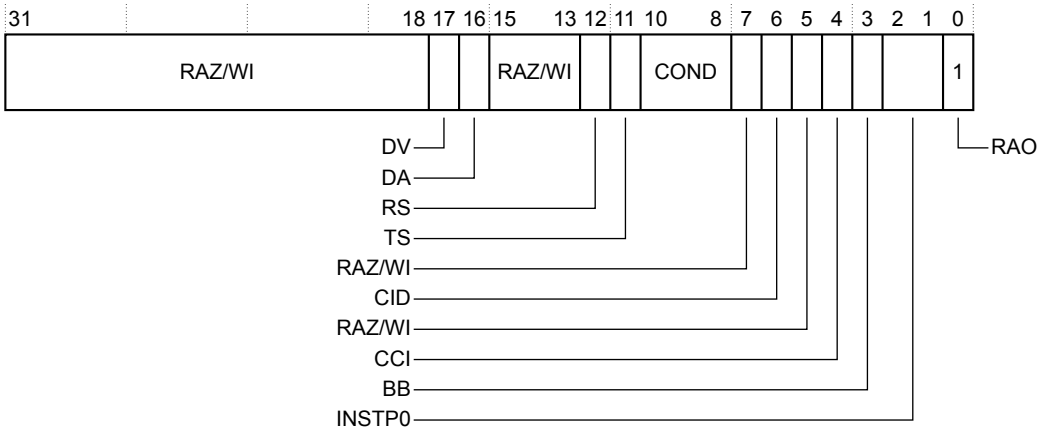


Figure 11-8 TRCCONFIGR bit assignments

The following table shows the TRCCONFIGR bit assignments.

Table 11-20 TRCCONFIGR bit assignments

Bits	Name	Function
[31:18]	-	Reserved. RAZ/WI.
[17]	DV	Data value tracing: `0b0` Data value tracing disabled. `0b1` Data value tracing enabled.
[16]	DA	Data address tracing: `0b0` Data address tracing disabled. `0b1` Data address tracing enabled.
[15:13]	-	Reserved. RAZ/WI.
[12]	RS	Return stack enable: `0b0` Return stack disabled. `0b1` Return stack enabled.
[11]	TS	Global timestamp tracing: `0b0` Global timestamp tracing disabled. `0b1` Global timestamp tracing enabled. For more global timestamping options, see *11.8.9 Global Timestamp Control Register* on page 11-276.
[10:8]	COND	Conditional instruction tracing: `0b000` Conditional instruction tracing disabled. `0b001` Conditional load instructions are traced. `0b010` Conditional store instructions are traced. `0b011` Conditional load and store instructions are traced. `0b111` All conditional instructions are traced. All other values are Reserved.

Table 11-20 TRCCONFIGR bit assignments (continued)

Bits	Name	Function
[7]	-	Reserved. RAZ/WI.
[6]	CID	Context ID tracing: `0b0` Context ID tracing disabled. `0b1` Context ID tracing enabled.
[5]	-	Reserved. RAZ/WI.
[4]	CCI	Cycle counting in instruction trace: `0b0` Cycle counting in instruction trace disabled. `0b1` Cycle counting in instruction trace. For more cycle counting options, see *11.8.11 Cycle Count Control Register* on page 11-278.
[3]	BB	Branch broadcast mode: `0b0` Branch broadcast mode disabled. `0b1` Branch broadcast mode trace. For more branch broadcast mode options, see *11.8.12 Branch Broadcast Control Register* on page 11-279.
[2:1]	INSTP0	Determines the instructions which are P0 instructions: `0b00` Only branches are P0 instructions. `0b01` Load instructions and branches are P0 instructions. `0b10` Store instructions and branches are P0 instructions. `0b11` Load and store instructions and branches are P0 instructions.
[0]	-	Reserved. RAO.

Related reference

11.7.1 Cortex®-R8 processor ETM register summary on page 11-254
General control and ID registers on page 11-258
11.8.9 Global Timestamp Control Register on page 11-276
11.8.11 Cycle Count Control Register on page 11-278
11.8.12 Branch Broadcast Control Register on page 11-279

11.8.5 Auxiliary Control Register

The TRCAUXCTLR provides additional controls for the Cortex-R8 processor ETM.

Usage constraints

There are no usage constraints.

Configurations

Available in all configurations.

Attributes

Register number: 6

Base offset `0x018`

Name: TRCAUXCTLR

Type: RW

Reset: `0x00000000`

The following figure shows the TRCAUXCTLR bit assignments.

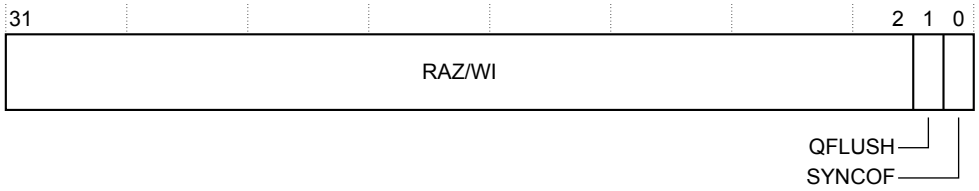


Figure 11-9 TRCAUXCTLR bit assignments

The following table shows the TRCAUXCTLR bit assignments.

Table 11-21 TRCAUXCTLR bit assignments

Bits	Name	Function
[31:2]	-	Reserved. RAZ/WI.
[1]	QFLUSH	Always respond immediately to TRCITDATBOUTR.AFREADY. No interaction with FIFO draining, even in WFI state.
[0]	SYNCOF	Force an overflow if synchronization is not completed when second synchronization is due.

Related reference

11.7.1 Cortex®-R8 processor ETM register summary on page 11-254

General control and ID registers on page 11-258

11.8.6 Event Control 0 Register

The TRCEVENTCTL0R controls the tracing of events in the trace stream. The events also drive the external outputs from the Cortex-R8 processor ETM.

Usage constraints

There are no usage constraints.

Configurations

Available in all configurations.

Attributes

Register number: 8

Base offset `0x020`

Name: TRCEVENTCTL0R

Type: RW

Reset: -

The following figure shows the TRCEVENTCTL0R bit assignments.

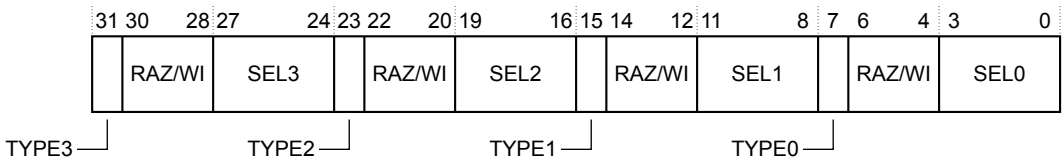


Figure 11-10 TRCEVENTCTL0R bit assignments

The following table shows the TRCEVENTCTL0R bit assignments.

Table 11-22 TRCEVENTCTL0R bit assignments

Bits	Name	Function
[31]	TYPE3	Selects the resource type for event 3: `0b0` Single selected resource. `0b1` Boolean combined resource pair.
[30:28]	-	Reserved. RAZ/WI.
[27:24]	SEL3	Selects the resource number, based on the value of TYPE3: When TYPE3 is `0b0`, selects a single selected resource from 0-15 defined by bits[3:0]. When TYPE3 is `0b1`, selects a Boolean combined resource pair from 0-7 defined by bits[2:0].
[23]	TYPE2	Selects the resource type for event 2: `0b0` Single selected resource. `0b1` Boolean combined resource pair.
[22:20]	-	Reserved. RAZ/WI.
[19:16]	SEL2	Selects the resource number, based on the value of TYPE2: When TYPE2 is `0b0`, selects a single selected resource from 0-15 defined by bits[3:0]. When TYPE2 is `0b1`, selects a Boolean combined resource pair from 0-7 defined by bits[2:0].
[15]	TYPE1	Selects the resource type for event 1: `0b0` Single selected resource. `0b1` Boolean combined resource pair.
[14:12]	-	Reserved. RAZ/WI.
[11:8]	SEL1	Selects the resource number, based on the value of TYPE1: When TYPE1 is `0b0`, selects a single selected resource from 0-15 defined by bits[3:0]. When TYPE1 is `0b1`, selects a Boolean combined resource pair from 0-7 defined by bits[2:0].
[7]	TYPE0	Selects the resource type for event 0: `0b0` Single selected resource. `0b1` Boolean combined resource pair.

Table 11-22 TRCEVENTCTL0R bit assignments (continued)

Bits	Name	Function
[6:4]	-	Reserved. RAZ/WI.
[3:0]	SEL0	Selects the resource number, based on the value of TYPE0: When TYPE0 is `0b0`, selects a single selected resource from 0-15 defined by bits[3:0]. When TYPE0 is `0b1`, selects a Boolean combined resource pair from 0-7 defined by bits[2:0].

Related reference
11.7.1 Cortex®-R8 processor ETM register summary on page 11-254
General control and ID registers on page 11-258

11.8.7 Event Control 1 Register

The TRCEVENTCTL1R controls how the events selected by TRCEVENTCTL0R behave.

Usage constraints
There are no usage constraints.

Configurations
Available in all configurations.

Attributes

Register number: 9

Base offset `0x024`

Name: TRCEVENTCTL1R

Type: RW

Reset: -

The following figure shows the TRCEVENTCTL1R bit assignments.

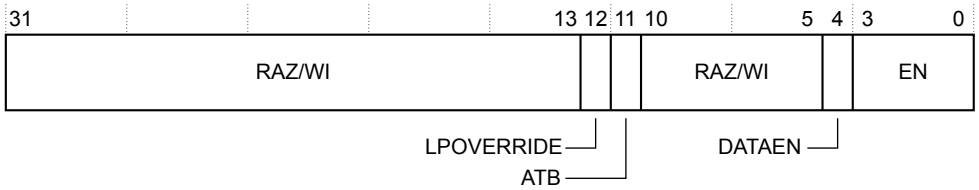


Figure 11-11 TRCEVENTCTL1R bit assignments

The following table shows the TRCEVENTCTL1R bit assignments.

Table 11-23 TRCEVENTCTL1R bit assignments

Bits	Name	Function
[31:13]	-	Reserved. RAZ/WI.
[12]	LPOVERRIDE	Low-power state behavior override: `0b0` Low-power state behavior unaffected. `0b1` Low-power state behavior overridden. The resources and Event trace generation are unaffected by entry to a low-power state.

Table 11-23 TRCEVENTCTL1R bit assignments (continued)

Bits	Name	Function
[11]	ATB	ATB trigger enable: `0b0` ATB trigger disabled. `0b1` ATB trigger enabled.
[10:5]	-	Reserved. RAZ/WI.
[4]	DATAEN	Enables generation of an event element in the data trace stream when the selected event occurs: `0b0` Event does not cause an event element. `0b1` Event causes an event element.
[3:0]	EN	One bit per event, to enable generation of an event element in the instruction trace stream when the selected event occurs: `0b0` Event does not cause an event element. `0b1` Event causes an event element.

Related reference

11.8.6 Event Control 0 Register on page 11-272

11.7.1 Cortex®-R8 processor ETM register summary on page 11-254

General control and ID registers on page 11-258

11.8.8 Stall Control Register

The TRCSTALLCTLR enables the Cortex-R8 processor ETM to stall the Cortex-R8 processor if the Cortex-R8 processor ETM FIFO overflows.

Usage constraints

There are no usage constraints.

Configurations

Available in all configurations.

Attributes

Register number: 11

Base offset `0x02C`

Name: TRCSTALLCTLR

Type: RW

Reset: -

The following figure shows the TRCSTALLCTLR bit assignments.

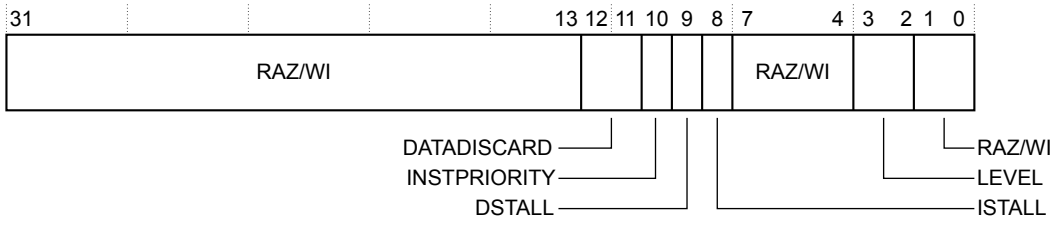


Figure 11-12 TRCSTALLCTLR bit assignments

The following table shows the TRCSTALLCTLR bit assignments.

Table 11-24 TRCSTALLCTLR bit assignments

Bits	Name	Function
[31:13]	-	Reserved. RAZ/WI.
[12:11]	DATADISCARD	Sets the priority of data trace components, enabling the Cortex-R8 processor ETM to discard some data if the data trace buffer space is less than LEVEL: `0b00` Discard no data. `0b01` Discard loaded data transfers. `0b10` Discard stored data transfers. `0b11` Discard both loaded and stored data transfers.
[10]	INSTPRIORITY	Prioritize instruction trace if instruction trace buffer space is less than LEVEL: `0b0` Do not prioritize instruction trace. `0b1` Prioritize instruction trace.
[9]	DSTALL	Stall Cortex-R8 processor based on data trace buffer space: `0b0` Do not stall processor. `0b1` Stall processor if data trace buffer space is less than LEVEL.
[8]	ISTALL	Stall Cortex-R8 processor based on instruction trace buffer space: `0b0` Do not stall processor. `0b1` Stall processor if instruction trace buffer space is less than LEVEL.
[7:4]	-	Reserved. RAZ/WI.
[3:2]	LEVEL	Threshold at which stalling becomes active. This provides four levels. This level can be varied to optimize the level of invasion caused by stalling, balanced against the risk of a FIFO overflow: `0b00` Lowest level, where zero invasion occurs. `0b11` Highest level, where the most invasion occurs to reduce the risk of overflow.
[1:0]	-	Reserved. RAZ/WI.

Related reference

11.7.1 Cortex®-R8 processor ETM register summary on page 11-254

General control and ID registers on page 11-258

11.8.9 Global Timestamp Control Register

The TRCTSCTLR controls the insertion of global timestamps into the trace streams. A timestamp is always inserted into the instruction trace stream, and also in the data trace stream if any data tracing is enabled.

Usage constraints

There are no usage constraints.

Configurations

Available in all configurations.

Attributes

Register number: 12

Base offset `0x030`

Name: TRCTSCTLR

Type: RW

Reset: -

The following figure shows the TRCTSCTLR bit assignments.

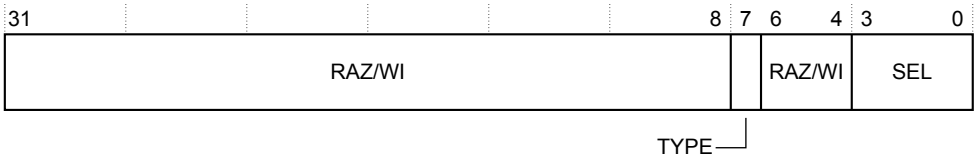


Figure 11-13 TRCTSCTLR bit assignments

The following table shows the TRCTSCTLR bit assignments.

Table 11-25 TRCTSCTLR bit assignments

Bits	Name	Function
[31:8]	-	Reserved. RAZ/WI.
[7]	TYPE	Selects the resource type: `0b0` Single selected resource. `0b1` Boolean combined resource pair.
[6:4]	-	Reserved. RAZ/WI
[3:0]	SEL	Selects the resource number, based on the value of TYPE: When TYPE is `0b0`, selects a single selected resource from 0-15 defined by bits[3:0]. When TYPE is `0b1`, selects a Boolean combined resource pair from 0-7 defined by bits[2:0].

Related reference

11.7.1 Cortex®-R8 processor ETM register summary on page 11-254

General control and ID registers on page 11-258

11.8.10 Synchronization Period Register

The TRCSYNCPR specifies the period of synchronization of the trace streams. TRCSYNCPR defines several bytes of trace between requests for synchronization. This value is always a power of two.

Usage constraints

There are no usage constraints.

Configurations

Available in all configurations.

Attributes

Register number: 13

Base offset `0x034`

Name: TRCSYNCPR

Type: RW

Reset: -

The following figure shows the TRCSYNCPR bit assignments.

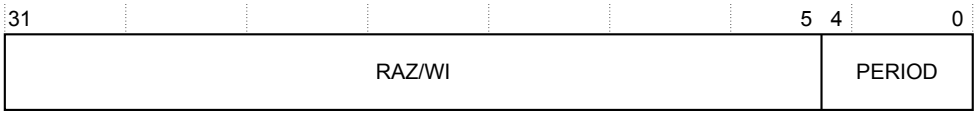


Figure 11-14 TRCSYNCPR bit assignments

The following table shows the TRCSYNCPR bit assignments.

Table 11-26 TRCSYNCPR bit assignments

Bits	Name	Function
[31:5]	-	Reserved. RAZ/WI.
[4:0]	PERIOD	Defines the number of bytes of trace between synchronization requests as a total of the number of bytes generated by both the instruction and data streams. The number of bytes is 2^N where N is the value of this field: • A value of zero disables these periodic synchronization requests, but does not disable other synchronization requests. • The minimum value that can be programmed, other than zero, is 8, providing a minimum synchronization period of 256 bytes. • The maximum value is 20, providing a maximum synchronization period of 2^{20} bytes.

Related reference

11.7.1 Cortex®-R8 processor ETM register summary on page 11-254

General control and ID registers on page 11-258

11.8.11 Cycle Count Control Register

The TRCCCCTLR sets the threshold value for cycle counting.

Usage constraints

Writing a value of all zeroes is UNPREDICTABLE when instruction trace cycle counting is enabled.

Configurations

Available in all configurations.

Attributes

Register number: 14

Base offset `0x038`

Name: TRCCCCTLR

Type: RW

Reset: -

The following figure shows the TRCCCCTLR bit assignments.

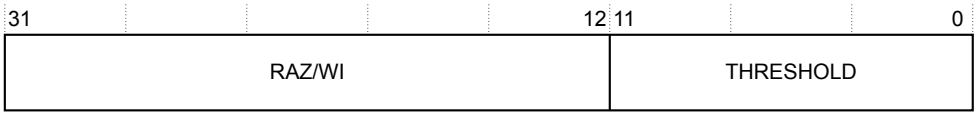


Figure 11-15 TRCCCCTLR bit assignments

The following table shows the TRCCCCTLR bit assignments.

Table 11-27 TRCCCCTLR bit assignments

Bits	Name	Function
[31:12]	-	Reserved. RAZ/WI
[11:0]	THRESHOLD	Instruction trace cycle count threshold

Related reference

11.7.1 Cortex®-R8 processor ETM register summary on page 11-254

General control and ID registers on page 11-258

11.8.12 Branch Broadcast Control Register

The TRCBBCTLR controls how branch broadcasting behaves, and enables branch broadcasting to be enabled for certain memory regions.

Usage constraints

There are no usage constraints.

Configurations

Available in all configurations.

Attributes

Register number: 15

Base offset `0x03C`

Name: TRCBBCTLR

Type: RW

Reset: -

The following figure shows the TRCBBCTLR bit assignments.

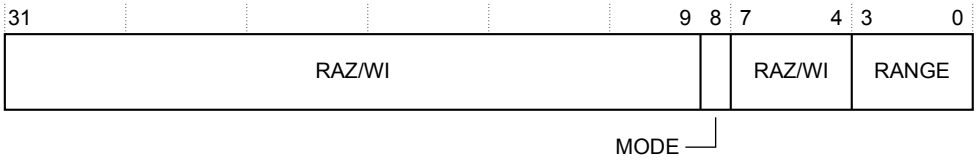


Figure 11-16 TRCBBCTLR bit assignments

The following table shows the TRCBBCTLR bit assignments.

Table 11-28 TRCBBCTLR bit assignments

Bits	Name	Function
[31:9]	-	Reserved. RAZ/WI.
[8]	MODE	Selects mode: `0b0` Exclude mode. The Address Range Comparators defined by the RANGE field indicate address ranges where branch broadcasting is not enabled. Selecting no ranges results in branch broadcasting being enabled over the whole memory map. `0b1` Include mode. The Address Range Comparators defined by the RANGE field indicate address ranges where branch broadcasting is enabled. Setting RANGE to all zeroes is UNPREDICTABLE when in Include mode.
[7:4]	-	Reserved. RAZ/WI.
[3:0]	RANGE	Selects Address Range Comparators to control where branch broadcasting is enabled. One bit is provided for each implemented Address Range Comparator.

Related reference

11.7.1 Cortex®-R8 processor ETM register summary on page 11-254

General control and ID registers on page 11-258

11.8.13 Trace ID Register

The TRCTRACEIDR sets the trace ID on the trace bus. Controls two trace IDs, one for instruction trace and one for data trace.

Usage constraints

In a CoreSight system, writing of reserved trace ID values, `0x00` and `0x70-0x7F`, is UNPREDICTABLE.

Configurations

Available in all configurations.

Attributes

Register number: 16

Base offset `0x040`

Name: TRCTRACEIDR

Type: RW

Reset: -

The following figure shows the TRCTRACEIDR bit assignments.

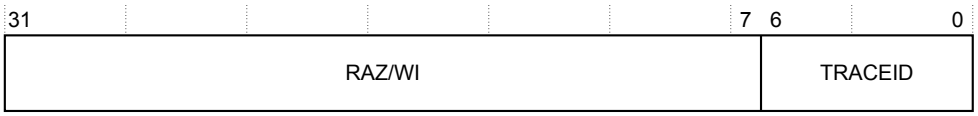


Figure 11-17 TRCTRACEIDR bit assignments

The following table shows the TRCTRACEIDR bit assignments.

Table 11-29 TRCTRACEIDR bit assignments

Bits	Name	Function
[31:7]	-	Reserved. RAZ/WI.
[6:0]	TRACEID	Trace ID value.When only instruction tracing is enabled, this provides the trace ID. When data tracing is enabled, this field must be written with bit[0] set to `0b0`. The instruction and data trace streams use adjacent trace ID values: • The instruction trace stream uses the trace ID {[6:1],0}. • The data value trace stream uses the trace ID {[6:1],1}. When data tracing is not enabled, bit[0] can be set to any value.

Related reference

11.7.1 Cortex®-R8 processor ETM register summary on page 11-254

Trace filtering control registers on page 11-258

11.8.14 ViewInst Main Control Register

The TRCVICTLR controls instruction trace filtering.

Usage constraints

There are no usage constraints.

Configurations

Available in all configurations.

Attributes

Register number: 32

Base offset `0x080`

Name: TRCVICTLR

Type: RW

Reset: -

The following figure shows the TRCVICTLR bit assignments.

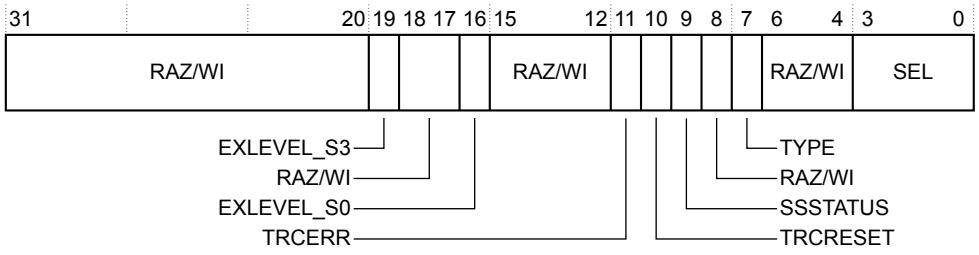


Figure 11-18 TRCVICTLR bit assignments

The following table shows the TRCVICTLR bit assignments.

Table 11-30 TRCVICTLR bit assignments

Bits	Value	Function
[31:20]	-	Reserved. RAZ/WI.
[19]	EXLEVEL_S3	Disables tracing in the specified exception level in Secure state for exception level 3: `0b0` Enable ViewInst in this exception level. `0b1` Disable ViewInst in this exception level.
[18:17]	-	Reserved. RAZ/WI
[16]	EXLEVEL_S0	Disables tracing in the specified exception level in Secure state for exception level 0: `0b0` Enable ViewInst in this exception level. `0b1` Disable ViewInst in this exception level.
[15:12]	-	Reserved. RAZ/WI.
[11]	TRCERR	Selects whether a system error exception must always be traced: `0b0` System error exception is traced only if the instruction or exception immediately before the system error exception is traced. `0b1` System error exception is always traced regardless of the value of ViewInst.
[10]	TRCRESET	Selects whether a reset exception must always be traced: `0b0` Reset exception is traced only if the instruction or exception immediately before the reset exception is traced. `0b1` Reset exception is always traced regardless of the value of ViewInst.
[9]	SSSTATUS	Indicates the current status of the start/stop logic: `0b0` Start/stop logic is in the stopped state. `0b1` Start/stop logic is in the started state.
[8]	-	Reserved. RAZ/WI.
[7]	TYPE	Selects the resource type: `0b0` Single selected resource. `0b1` Boolean combined resource pair.
[6:4]	-	Reserved. RAZ/WI.
[3:0]	SEL	Selects the resource number, based on the value of TYPE: When TYPE is `0b0`, selects a single selected resource from 0-15 defined by bits[3:0]. When TYPE is `0b1`, selects a Boolean combined resource pair from 0-7 defined by bits[2:0].

Related reference

11.7.1 Cortex®-R8 processor ETM register summary on page 11-254

Trace filtering control registers on page 11-258

11.8.15 ViewInst Include/Exclude Control Register

The TRCVIIECTLR defines the address range comparators that control the ViewInst Include/Exclude control.

Usage constraints

Can only be written when the Cortex-R8 processor ETM is disabled.

Configurations

Available in all configurations.

Attributes

Register number: 33

Base offset `0x084`

Name: TRCVIIECTLR

Type: RW

Reset: -

The following figure shows the TRCVIIECTLR bit assignments.

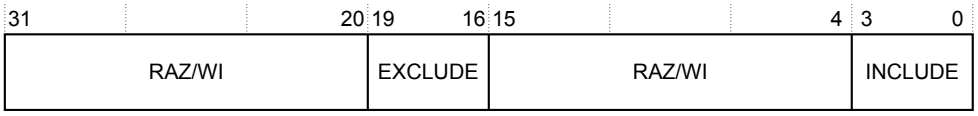


Figure 11-19 TRCVIIECTLR bit assignments

The following table shows the TRCVIIECTLR bit assignments.

Table 11-31 TRCVIIECTLR bit assignments

Bits	Name	Function
[31:20]	-	Reserved. RAZ/WI.
[19:16]	EXCLUDE	Defines the address range comparators for ViewInst exclude control. One bit is provided for each implemented Address Range Comparator.
[15:4]	-	Reserved. RAZ/WI.
[3:0]	INCLUDE	Defines the address range comparators for ViewInst include control. Selecting no include comparators indicates that all instructions must be included. The exclude control indicates which ranges must be excluded. One bit is provided for each implemented Address Range Comparator.

Related reference

11.7.1 Cortex®-R8 processor ETM register summary on page 11-254

Trace filtering control registers on page 11-258

11.8.16 ViewInst Start/Stop Control Register

The TRCVISSCTLR defines the single address comparators that control the ViewInst Start/Stop logic.

Usage constraints

Can only be written when the Cortex-R8 processor ETM is disabled.

Configurations

Available in all configurations.

Attributes

Register number: 34

Base offset `0x088`

Name: TRCVISSCTLR

Type: RW

Reset: -

The following figure shows the TRCVISSCTLR bit assignments.

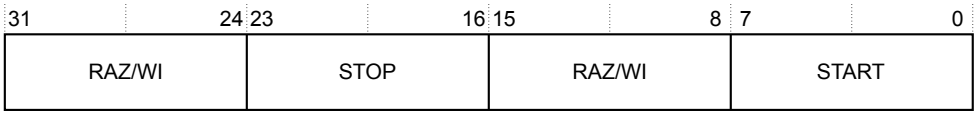


Figure 11-20 TRCVISSCTLR bit assignments

The following table shows the TRCVISSCTLR bit assignments.

Table 11-32 TRCVISSCTLR bit assignments

Bits	Name	Function
[31:24]	-	Reserved. RAZ/WI.
[23:16]	STOP	Defines the single address comparators to stop trace with the ViewInst Start/Stop control. One bit is provided for each implemented single address comparator.
[15:8]	-	Reserved. RAZ/WI.
[7:0]	START	Defines the single address comparators to start trace with the ViewInst Start/Stop control. One bit is provided for each implemented single address comparator.

Related reference

11.7.1 Cortex®-R8 processor ETM register summary on page 11-254

Trace filtering control registers on page 11-258

11.8.17 ViewData Main Control Register

The TRCVDCTLR controls data trace filtering.

Usage constraints

Can only be written when the Cortex-R8 processor ETM is disabled.

Configurations

Available in all configurations.

Attributes

Register number: 40

Base offset `0x0A0`

Name: TRCVDCTLR

Type: RW

Reset: -

The following figure shows the TRCVDCTLR bit assignments.

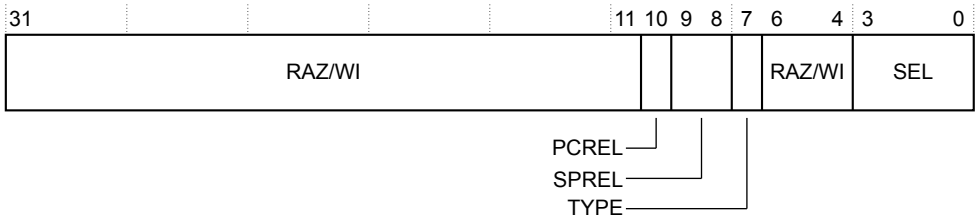


Figure 11-21 TRCVDCTLR bit assignments

The following table shows the TRCVDCTLR bit assignments.

Table 11-33 TRCVDCTLR bit assignments

Bits	Name	Function
[31:11]	-	Reserved. RAZ/WI.
[10]	PCREL	Controls tracing of data for transfers that are relative to the *Program Counter* (PC): `0b0` Tracing of PC-relative transfers is unaffected. `0b1` Do not trace either the address or value portions of PC-relative transfers.
[9:8]	SPREL	Controls tracing of data for transfers that are relative to the *Stack Pointer* (SP): `0b00` Tracing of SP-relative transfers is unaffected. `0b01` Reserved. `0b10` Do not trace the address portion of SP-relative transfers. A P1 data address element is generated if data value tracing is enabled. `0b11` Do not trace either the address or value portions of SP-relative transfers.
[7]	TYPE	Selects the resource type: `0b0` Single selected resource. `0b1` Boolean combined resource pair.
[6:4]	-	Reserved. RAZ/WI.
[3:0]	SEL	Selects the resource number, based on the value of TYPE: When TYPE is `0b0`, selects a single selected resource from 0-15 defined by bits[3:0]. When TYPE is `0b1`, selects a Boolean combined resource pair from 0-7 defined by bits[2:0].

Related reference

11.7.1 Cortex®-R8 processor ETM register summary on page 11-254

Trace filtering control registers on page 11-258

11.8.18 ViewData Include/Exclude Single Address Comparator Register

The TRCVDSACCTLR defines the single address comparators that control the ViewData Include/Exclude control.

Usage constraints

Can only be written when the Cortex-R8 processor ETM is disabled.

Configurations

Available in all configurations.

Attributes

Register number: 41

Base offset `0x0A4`

Name: TRCVDSACCTLR

Type: RW

Reset: -

The following figure shows the TRCVDSACCTLR bit assignments.

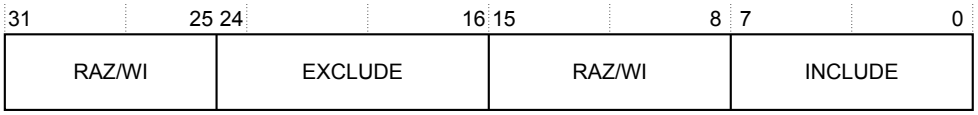


Figure 11-22 TRCVDSACCTLR bit assignments

The following table shows the TRCVDSACCTLR bit assignments.

Table 11-34 TRCVDSACCTLR bit assignments

Bits	Name	Function
[31:25]	-	Reserved. RAZ/WI.
[24:16]	EXCLUDE	Defines the single address comparators for ViewData exclude control. One bit is provided for each implemented address comparator.
[15:8]	-	Reserved. RAZ/WI.
[7:0]	INCLUDE	Defines the single address comparators for ViewData include control. One bit is provided for each implemented address comparator.

Related reference

11.7.1 Cortex®-R8 processor ETM register summary on page 11-254

Trace filtering control registers on page 11-258

11.8.19 ViewData Include/Exclude Address Range Comparator Register

The TRCVDARCCTLR defines the address range comparators that control the ViewData Include/Exclude control.

Usage constraints

Can only be written when the Cortex-R8 processor ETM is disabled.

Configurations

Available in all configurations.

Attributes

Register number: 42

Base offset `0x0A8`

Name: TRCVDARCCTLR

Type: RW

Reset: -

The following figure shows the TRCVDARCCTLR bit assignments.

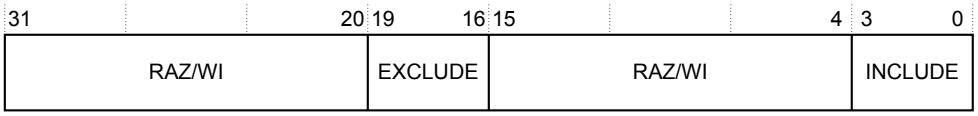


Figure 11-23 TRCVDARCCTLR bit assignments

The following table shows the TRCVDARCCTLR bit assignments.

Table 11-35 TRCVDARCCTLR bit assignments

Bits	Name	Function
[31:20]	-	Reserved. RAZ/WI.
[19:16]	EXCLUDE	Defines the address range comparators for ViewData exclude control.One bit is provided for each implemented address range comparator.
[15:4]	-	Reserved. RAZ/WI.
[3:0]	INCLUDE	Defines the address range comparators for ViewData include control. One bit is provided for each implemented address range comparator.

Related reference

11.7.1 Cortex®-R8 processor ETM register summary on page 11-254

Trace filtering control registers on page 11-258

11.8.20 Sequencer State Transition Control Registers 0-2

The TRCSEQEVRn define the sequencer transitions that progress to the next state or backwards to the previous state. The Cortex-R8 processor ETM implements a sequencer state machine with up to four states.

Usage constraints

Can only be written when the Cortex-R8 processor ETM is disabled.

Configurations

Available in all configurations.

Attributes

Register number: 64-66

Base offset `0x100-0x108`

Name: TRCSEQEVRn

Type: RW

Reset: -

The following figure shows the TRCSEQEVRn bit assignments.

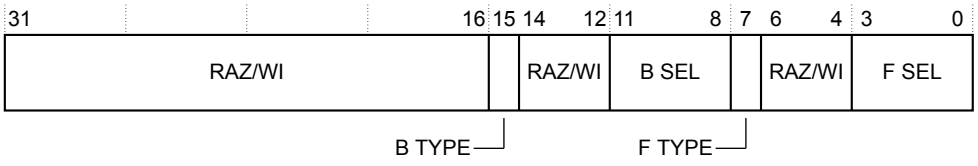


Figure 11-24 TRCSEQEVRn bit assignments

The following table shows the TRCSEQEVRn bit assignments.

Table 11-36 TRCSEQEVRn bit assignments

Bits	Name	Function
[31:16]	-	Reserved. RAZ/WI.
[15]	B TYPE	Selects the resource type to move backwards to this state from the next state: `0b0` Single selected resource. `0b1` Boolean combined resource pair.
[14:12]	-	Reserved. RAZ/WI.
[11:8]	B SEL	Selects the resource number, based on the value of B TYPE: When B TYPE is `0b0`, selects a single selected resource from 0-15 defined by bits[3:0]. When B TYPE is `0b1`, selects a Boolean combined resource pair from 0-7 defined by bits[2:0].
[7]	F TYPE	Selects the resource type to move forwards from this state to the next state: `0b0` Single selected resource. `0b1` Boolean combined resource pair.
[6:4]	-	Reserved. RAZ/WI.
[3:0]	F SEL	Selects the resource number, based on the value of F TYPE: When F TYPE is `0b0`, selects a single selected resource from 0-15 defined by bits[3:0]. When F TYPE is `0b1`, selects a Boolean combined resource pair from 0-7 defined by bits[2:0].

Related reference
11.7.1 Cortex®-R8 processor ETM register summary on page 11-254
Derived resource registers on page 11-259

11.8.21 Sequencer Reset Control Register

The TRCSEQRSTEVR resets the sequencer to state 0.

Usage constraints
Can only be written when the Cortex-R8 processor ETM is disabled.

Configurations
Available in all configurations.

Attributes

Register number: 70

Base offset `0x118`

Name: TRCSEQRSTEVR

Type: RW

Reset: -

The following figure shows the TRCSEQRSTEVR bit assignments.

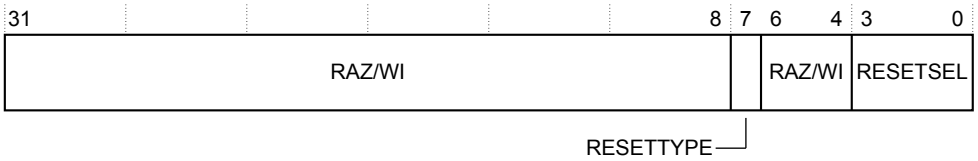


Figure 11-25 TRCSEQRSTEVR bit assignments

The following table shows the TRCSEQRSTEVR bit assignments.

Table 11-37 TRCSEQRSTEVR bit assignments

Bits	Name	Function
[31:8]	-	Reserved. RAZ/WI.
[7]	RESETTYPE	Selects the resource type to move back to state 0: `0b0` Single selected resource. `0b1` Boolean combined resource pair.
[6:4]	-	Reserved. RAZ/WI.
[3:0]	RESETSEL	Selects the resource number, based on the value of RESETTYPE: When RESETTYPE is `0b0`, selects a single selected resource from 0-15 defined by bits[3:0]. When RESETTYPE is `0b1`, selects a Boolean combined resource pair from 0-7 defined by bits[2:0].

Related reference
11.7.1 Cortex®-R8 processor ETM register summary on page 11-254
Derived resource registers on page 11-259

11.8.22 Sequencer State Register

The TRCSEQSTR holds the value of the current state of the sequencer.

Usage constraints

- Can only be written when the Cortex-R8 processor ETM is disabled.
- Must be programmed with an initial value when programming the sequencer.

Configurations

Available in all configurations.

Attributes

Register number: 71

Base offset `0x11C`

Name: TRCSEQSTR

Type: RW

Reset: -

The following figure shows the TRCSEQSTR bit assignments.

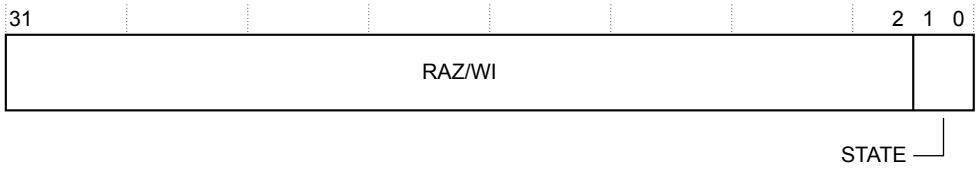


Figure 11-26 TRCSEQSTR bit assignments

The following table shows the TRCSEQSTR bit assignments.

Table 11-38 TRCSEQSTR bit assignments

Bits	Name	Function
[31:2]	-	Reserved. RAZ/WI
[1:0]	STATE	Current sequencer state: `0b00` State 0 `0b01` State 1 `0b10` State 2 `0b11` State 3

Related reference

11.7.1 Cortex®-R8 processor ETM register summary on page 11-254

Derived resource registers on page 11-259

11.8.23 External Input Select Register

The TRCEXTINSELR controls the selectors that choose an external input as a resource in the Cortex-R8 processor ETM.

Usage constraints

Can only be written when the Cortex-R8 processor ETM is disabled.

Configurations

Available in all configurations.

Attributes

Register number: 72

Base offset `0x120`

Name: TRCEXTINSELR

Type: RW

Reset: -

The following figure shows the TRCEXTINSELR bit assignments.

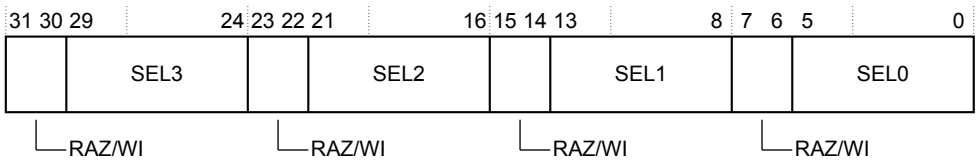


Figure 11-27 TRCEXTINSELR bit assignments

The following table shows the TRCEXTINSELR bit assignments.

Table 11-39 TRCEXTINSELR bit assignments

Bits	Name	Function
[31:30]	-	Reserved. RAZ/WI
[29:24]	SEL3	Selects an event from the external input bus for External Input Resource 3
[23:22]	-	Reserved. RAZ/WI
[21:16]	SEL2	Selects an event from the external input bus for External Input Resource 2
[15:14]	-	Reserved. RAZ/WI
[13:8]	SEL1	Selects an event from the external input bus for External Input Resource 1
[7:6]	-	Reserved. RAZ/WI
[5:0]	SEL0	Selects an event from the external input bus for External Input Resource 0

Related reference

11.7.1 Cortex®-R8 processor ETM register summary on page 11-254

Derived resource registers on page 11-259

11.8.24 Counter Reload Value Registers 0-1

The TRCCNTRLDVRn define the reload value for the counter.

Usage constraints

Can only be written when the Cortex-R8 processor ETM is disabled.

Configurations

Available in all configurations.

Attributes

Register number: 80-81

Base offset `0x140-0x144`

Name: TRCCNTRLDVRn

Type: RW

Reset: -

The following figure shows the TRCCNTRLDVRn bit assignments.

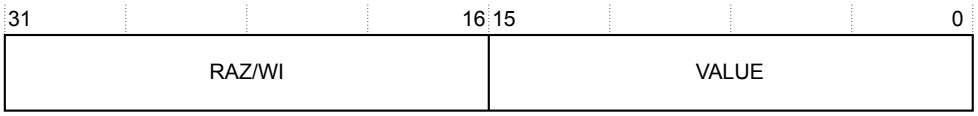


Figure 11-28 TRCCNTRLDVRn bit assignments

The following table shows the TRCCNTRLDVRn bit assignments.

Table 11-40 TRCCNTRLDVRn bit assignments

Bits	Value	Function
[31:16]	-	Reserved. RAZ/WI.
[15:0]	VALUE	Defines the reload value for the counter. This value is loaded into the counter each time the reload event occurs.

Related reference

11.7.1 Cortex®-R8 processor ETM register summary on page 11-254

Derived resource registers on page 11-259

11.8.25 Counter Control Register 0

The TRCCNTCTLR0 controls the counter.

Usage constraints

Can only be written when the Cortex-R8 processor ETM is disabled.

Configurations

Available in all configurations.

Attributes

Register number: 84

Base offset `0x150`

Name: TRCCNTCTLR0

Type: RW

Reset: -

The following figure shows the TRCCNTCTLR0 bit assignments.

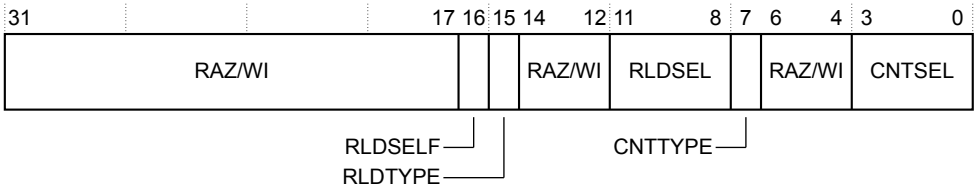


Figure 11-29 TRCCNTCTLR0 bit assignments

The following table shows the TRCCNTCTLR0 bit assignments.

Table 11-41 TRCCNTCTLR0 bit assignments

Bits	Name	Function
[31:17]	-	Reserved. RAZ/WI.
[16]	RLDSELF	Defines whether the counter reloads when it reaches zero: `0b0` The counter does not reload when it reaches zero. The counter only reloads based on RLDTYPE and RLDSEL. `0b1` The counter reloads when it reaches zero and the resource selected by CNTTYPE and CNTSEL is also active. The counter also reloads based on RLDTYPE and RLDSEL.
[15]	RLDTYPE	Selects the resource type for the reload: `0b0` Single selected resource. `0b1` Boolean combined resource pair.
[14:12]	-	Reserved. RAZ/WI.
[11:8]	RLDSEL	Selects the resource number, based on the value of RLDTYPE: When RLDTYPE is `0b0`, selects a single selected resource from 0-15 defined by bits[3:0]. When RLDTYPE is `0b1`, selects a Boolean combined resource pair from 0-7 defined by bits[2:0].

Table 11-41 TRCCNTCTLR0 bit assignments (continued)

Bits	Name	Function
[7]	CNTTYPE	Selects the resource type for the counter: `0b0` Single selected resource. `0b1` Boolean combined resource pair.
[6:4]	-	Reserved. RAZ/WI.
[3:0]	CNTSEL	Selects the resource number, based on the value of CNTTYPE: When CNTTYPE is `0b0`, selects a single selected resource from 0-15 defined by bits[3:0]. When CNTTYPE is `0b1`, selects a Boolean combined resource pair from 0-7 defined by bits[2:0].

Related reference
11.7.1 Cortex®-R8 processor ETM register summary on page 11-254
Derived resource registers on page 11-259

11.8.26 Counter Control Register 1

The TRCCNTCTLR1 controls the counter.

Usage constraints
Can only be written when the Cortex-R8 processor ETM is disabled.

Configurations
Available in all configurations.

Attributes

Register number: 85

Base offset `0x154`

Name: TRCCNTCTLR1

Type: RW

Reset: -

The following figure shows the TRCCNTCTLR1 bit assignments.

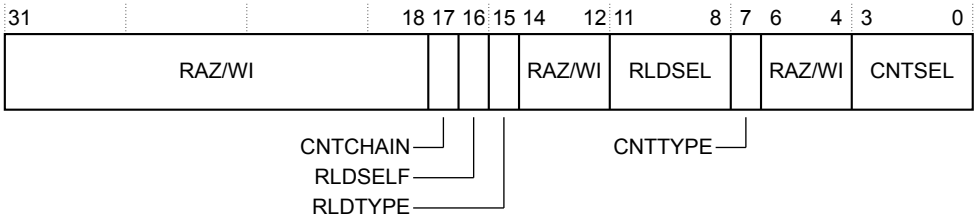


Figure 11-30 TRCCNTCTLR1 bit assignments

The following table shows the TRCCNTCTLR1 bit assignments.

Table 11-42 TRCCNTCTLR1 bit assignments

Bits	Name	Function
[31:18]	-	Reserved. RAZ/WI.
[17]	CNTCHAIN	Defines whether counter1 decrements when counter0 reloads. This enables two counters to be used in combination to provide a larger counter: `0b0` Counter0 operates independently of counter1. The counter only decrements based on CNTTYPE and CNTSEL. `0b1` Counter1 decrements when counter0 reloads. The counter also decrements when the resource selected by CNTTYPE and CNTSEL is active.
[16]	RLDSELF	Defines whether the counter reloads when it reaches zero: `0b0` The counter does not reload when it reaches zero. The counter only reloads based on RLDTYPE and RLDSEL. `0b1` The counter reloads when it is zero and the resource selected by CNTTYPE and CNTSEL is also active. The counter also reloads based on RLDTYPE and RLDSEL.
[15]	RLDTYPE	Selects the resource type for the reload: `0b0` Single selected resource. `0b1` Boolean combined resource pair.
[14:12]	-	Reserved. RAZ/WI.
[11:8]	RLDSEL	Selects the resource number, based on the value of RLDTYPE: When RLDTYPE is `0b0`, selects a single selected resource from 0-15 defined by bits[3:0]. When RLDTYPE is `0b1`, selects a Boolean combined resource pair from 0-7 defined by bits[2:0].
[7]	CNTTYPE	Selects the resource type for the counter: `0b0` Single selected resource. `0b1` Boolean combined resource pair.
[6:4]	-	Reserved. RAZ/WI.
[3:0]	CNTSEL	Selects the resource number, based on the value of CNTTYPE: When CNTTYPE is `0b0`, selects a single selected resource from 0-15 defined by bits[3:0]. When CNTTYPE is `0b1`, selects a Boolean combined resource pair from 0-7 defined by bits[2:0].

Related reference

11.7.1 Cortex®-R8 processor ETM register summary on page 11-254

Derived resource registers on page 11-259

11.8.27 Counter Value Registers 0-1

The TRCCNTVRn contain the current counter value.

Usage constraints

- Can only be written when the Cortex-R8 processor ETM is disabled.
- Must be programmed with an initial value when programming the counter.

Configurations

Available in all configurations.

Attributes

Register number: 88-89

Base offset `0x160-0x164`

Name: TRCCNTVRn

Type: RW

Reset: -

The following figure shows the TRCCNTVRn bit assignments.

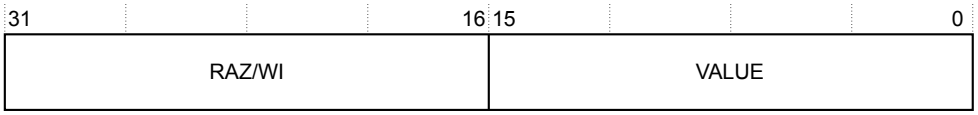


Figure 11-31 TRCCNTVRn bit assignments

The following table shows the TRCCNTVRn bit assignments.

Table 11-43 TRCCNTVRn bit assignments

Bits	Value	Function
[31:16]	-	Reserved. RAZ/WI
[15:0]	VALUE	Contains the current counter value

Related reference

11.7.1 Cortex®-R8 processor ETM register summary on page 11-254

Derived resource registers on page 11-259

11.8.28 ID Register 8-13

The TRCIDR8-13 indicate information about the trace stream that is required to analyze the trace.

Usage constraints

There are no usage constraints.

Configurations

Available in all configurations.

Attributes

Register number: 96

Base offset `0x180`

Name: TRCIDR8

Type: RO

Reset: `0x00000040`

The following figure shows the TRCIDR8 bit assignments.

Figure 11-32 TRCIDR8 bit assignments

The following table shows the TRCIDR8 bit assignments.

Table 11-44 TRCIDR8 bit assignments

Bits	Name	Function
[31:0]	MAXSPEC	Indicates the maximum speculation depth of the instruction trace stream. This is the maximum number of P0 elements that have not been committed in the trace stream at any one time. This field reads as `0x00000040` (64).

The following figure shows the TRCIDR9 bit assignments.

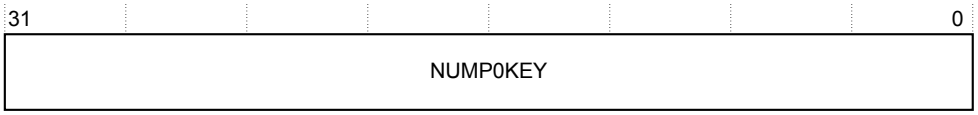


Figure 11-33 TRCIDR9 bit assignments

The following table shows the TRCIDR9 bit assignments.

Table 11-45 TRCIDR9 bit assignments

Bits	Name	Function
[31:0]	NUMP0KEY	Indicates the number of P0 right-hand keys that are used. This field reads as `0x00000040` (64).

The following figure shows the TRCIDR10 bit assignments.

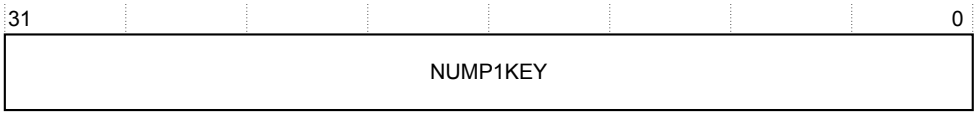


Figure 11-34 TRCIDR10 bit assignments

The following table shows the TRCIDR10 bit assignments.

Table 11-46 TRCIDR10 bit assignments

Bits	Name	Function
[31:0]	NUMP1KEY	Indicates the total number of P1 right-hand keys, including normal and special keys. This field reads as `0x00000040` (64 keys).

The following figure shows the TRCIDR11 bit assignments.

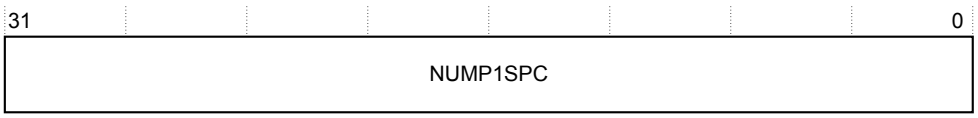


Figure 11-35 TRCIDR11 bit assignments

The following table shows the TRCIDR11 bit assignments.

Table 11-47 TRCIDR11 bit assignments

Bits	Name	Function
[31:0]	NUMP1SPC	Indicates the number of special P1 right-hand keys. This field reads as `0x00000011` (17 keys).

The following figure shows the TRCIDR12 bit assignments.

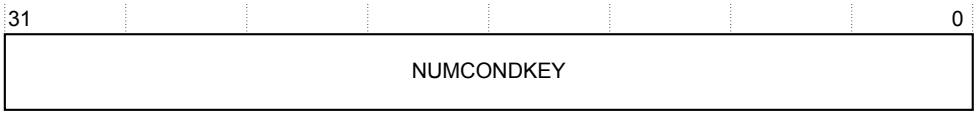


Figure 11-36 TRCIDR12 bit assignments

The following table shows the TRCIDR12 bit assignments.

Table 11-48 TRCIDR12 bit assignments

Bits	Name	Function
[31:0]	NUMCONDKEY	Indicates the total number of conditional instruction right-hand keys, including normal and special keys. This field reads as `0x00000020` (32).

The following figure shows the TRCIDR13 bit assignments.

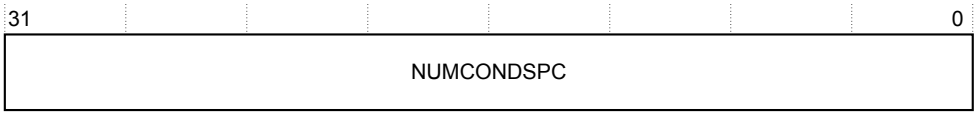


Figure 11-37 TRCIDR13 bit assignments

The following table shows the TRCIDR13 bit assignments.

Table 11-49 TRCIDR13 bit assignments

Bits	Name	Function
[31:0]	NUMCONDSPC	This indicates the number of special conditional instruction right-hand keys. There are no special conditional keys, so this field reads as `0x0`.

Related reference

11.7.1 Cortex®-R8 processor ETM register summary on page 11-254

Implementation-specific and identification registers on page 11-260

11.8.29 Implementation Specific Register 0

The TRCIMSPEC0 shows the presence of any implementation-specific features, and enables any features that are provided.

Usage constraints

There are no usage constraints.

Configurations

Available in all configurations.

Attributes

Register number: 112

Base offset `0x1C0`

Name: TRCIMSPEC0

Type: RW

Reset: `0x00000000`

The following figure shows the TRCIMSPEC0 bit assignments.

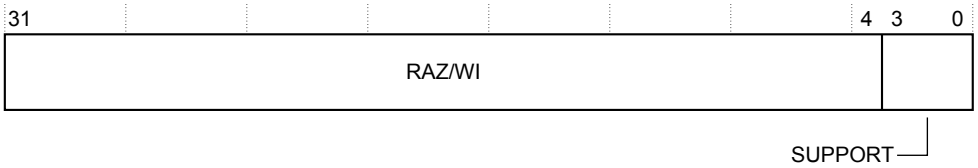


Figure 11-38 TRCIMSPEC0 bit assignments

The following table shows the TRCIMSPEC0 bit assignments.

Table 11-50 TRCIMSPEC0 bit assignments

Bits	Name	Function
[31:4]	-	Reserved. RAZ/WI.
[3:0]	SUPPORT	Set to `0x0`. No implementation-specific extensions are supported.

Related reference

11.7.1 Cortex®-R8 processor ETM register summary on page 11-254

Implementation-specific and identification registers on page 11-260

11.8.30 ID Register 0

The TRCIDR0 indicates the tracing capabilities of the Cortex-R8 processor ETM.

Usage constraints

There are no usage constraints.

Configurations

Available in all configurations.

Attributes

Register number: 120

Base offset `0x1E0`

Name: TRCIDR0

Type: RO

Reset: `0xXX001EFF`

The following figure shows the TRCIDR0 bit assignments.

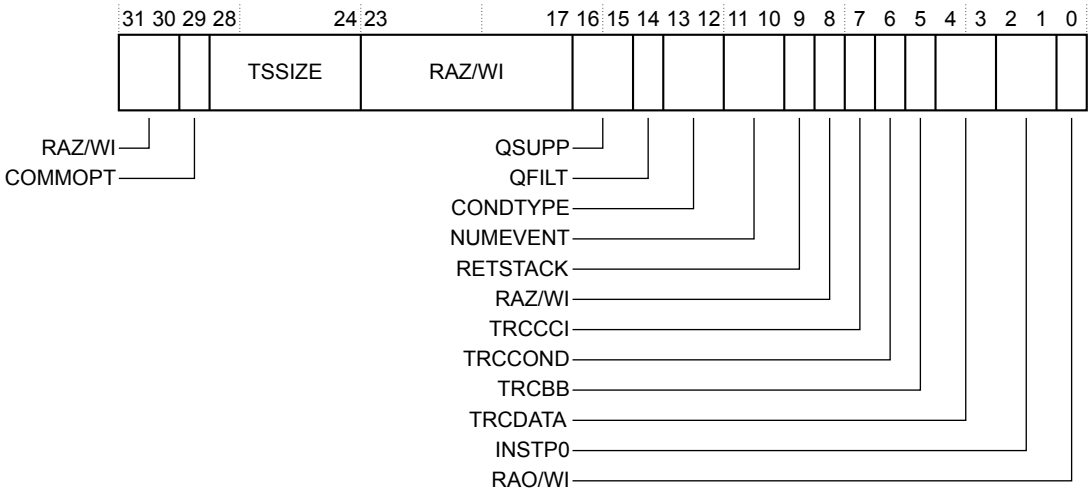


Figure 11-39 TRCIDR0 bit assignments

The following table shows the TRCIDR0 bit assignments.

Table 11-51 TRCIDR0 bit assignments

Bits	Name	Function
[31:30]	-	Reserved. RAZ/WI.
[29]	COMMOPT	Indicates the meaning of the commit field in some packets: `0b0` Commit mode 0.
[28:24]	TSSIZE	Global timestamp size. Driven from external **TSSIZE** pin: `0b00110` Maximum of 48-bit global timestamp implemented. **TSSIZE** is LOW. `0b01000` Maximum of 64-bit global timestamp implemented. **TSSIZE** is HIGH. Other values are Reserved.
[23:17]	-	Reserved. RAZ/WI.
[16:15]	QSUPP	Indicates Q element support: `0b00` Q elements not supported.
[14]	QFILT	Indicates Q element filtering support: `0b0` Q element filtering not supported.
[13:12]	CONDTYPE	Indicates how conditional results are traced: `0b01` Full CPSR traced.
[11:10]	NUMEVENT	Number of events supported in the trace, minus 1: **`0b11`** Four events supported.
[9]	RETSTACK	Return stack support: **`0b1`** Return stack implemented.

Table 11-51 TRCIDR0 bit assignments (continued)

Bits	Name	Function
[8]	-	Reserved. RAZ/WI.
[7]	TRCCCI	Support for cycle counting in the instruction trace: `0b1` Cycle counting in the instruction trace is implemented.
[6]	TRCCOND	Support for conditional instruction tracing: `0b1` Conditional instruction tracing is implemented.
[5]	TRCBB	Support for branch broadcast tracing: `0b1` Branch broadcast tracing is implemented.
[4:3]	TRCDATA	Support for tracing of data: `0b11` Tracing of data addresses and data values is implemented.
[2:1]	INSTP0	Support for tracing of load and store instructions as P0 elements: `0b11` Tracing of load and store instructions as P0 elements is implemented.
[0]	-	Reserved. RAO/WI.

Related reference
11.7.1 Cortex®-R8 processor ETM register summary on page 11-254
Implementation-specific and identification registers on page 11-260

11.8.31 ID Register 1

The TRCIDR1 indicates the basic architecture of the Cortex-R8 processor ETM.

Usage constraints
There are no usage constraints.

Configurations
Available in all configurations.

Attributes

Register number: 121

Base offset `0x1E4`

Name: TRCIDR1

Type: RO

Reset: `0x4100F400`

The following figure shows the TRCIDR1 bit assignments.

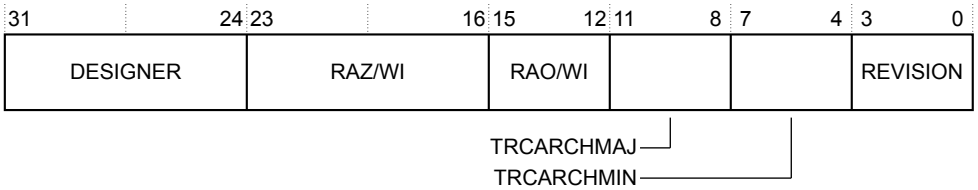


Figure 11-40 TRCIDR1 bit assignments

The following table shows the TRCIDR1 bit assignments.

Table 11-52 TRCIDR1 bit assignments

Bits	Name	Function
[31:24]	DESIGNER	Indicates the designer of the trace unit: `0x41` ASCII code for A, indicating Arm.
[23:16]	-	Reserved. RAZ/WI.
[15:12]	-	Reserved. RAO/WI.
[11:8]	TRCARCHMAJ	Major trace unit architecture version number: `0b0100` ETMv4.
[7:4]	TRCARCHMIN	Minor trace unit architecture version number: `0b0000` Minor revision 0.
[3:0]	REVISION	Implementation revision number: `0b0000` Implementation revision 0.

Related reference
11.7.1 Cortex®-R8 processor ETM register summary on page 11-254
Implementation-specific and identification registers on page 11-260

11.8.32 ID Register 2

The TRCIDR2 indicates the maximum sizes of certain aspects of items in the trace.

Usage constraints
There are no usage constraints.

Configurations
Available in all configurations.

Attributes

Register number: 122

Base offset `0x1E8`

Name: TRCIDR2

Type: RO

Reset: `0x00420084`

The following figure shows the TRCIDR2 bit assignments.

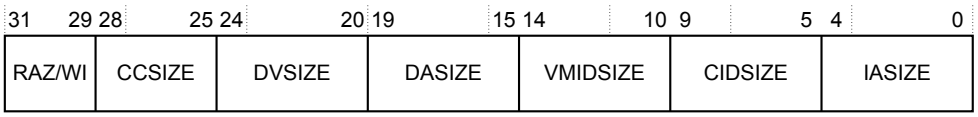


Figure 11-41 TRCIDR2 bit assignments

The following table shows the TRCIDR2 bit assignments.

Table 11-53 TRCIDR2 bit assignments

Bits	Name	Function
[31:29]	-	Reserved. RAZ/WI.
[28:25]	CCSIZE	Indicates the size of the cycle counter in bits minus 12: `0b0000` Cycle count is 12 bits.
[24:20]	DVSIZE	Data value size in bytes: `0b00100` Maximum of 32-bit data value size.
[19:15]	DASIZE	Data address size in bytes: `0b00100` Maximum of 32-bit address size.
[14:10]	VMIDSIZE	Virtual Machine ID size: `0b00000` Virtual Machine ID tracing not implemented.
[9:5]	CIDSIZE	Context ID size in bytes: `0b00100` Maximum of 32-bit Context ID size.
[4:0]	IASIZE	Instruction address size in bytes: `0b00100` Maximum of 32-bit address size.

Related reference
11.7.1 Cortex®-R8 processor ETM register summary on page 11-254
Implementation-specific and identification registers on page 11-260

11.8.33 ID Register 3

The TRCIDR3 indicates certain aspects of the Cortex-R8 processor ETM configuration.

Usage constraints
There are no usage constraints.

Configurations
Available in all configurations.

Attributes

Register number: 123

Base offset `0x1EC`

Name: TRCIDR3

Type: RO

Reset: `0xXX090004`

The following figure shows the TRCIDR3 bit assignments.

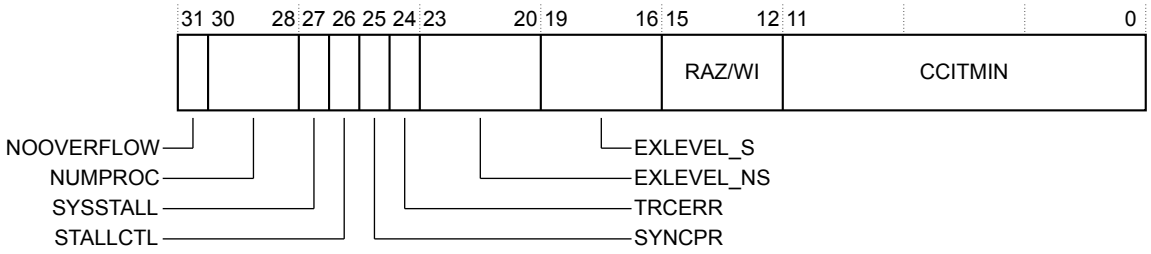


Figure 11-42 TRCIDR3 bit assignments

The following table shows the TRCIDR3 bit assignments.

Table 11-54 TRCIDR3 bit assignments

Bits	Name	Function
[31]	NOOVERFLOW	Indicates whether TRCSTALLCTLR.NOOVERFLOW is implemented: `0b0` NOOVERFLOW is not implemented.
[30:28]	NUMPROC	Number of cores available for tracing minus 1, indicating 1-8 cores. This describes the largest valid value which can be written to TRCPROCSELR.
[27]	SYSSTALL	System support for stall control of the Cortex-R8 processor. This is driven from the ETM **SYSSTALL** input pin, reflecting the system implementation: `0b0` System does not support stall control of the processor. `0b1` System supports stall control of the processor. This field is used with STALLCTL. Only when both SYSSTALL and STALLCTL are `0b1` does the system support stalling of the processor.
[26]	STALLCTL	Stall control support: `0b1` TRCSTALLCTLR is implemented. This field is used with SYSSTALL.
[25]	SYNCPR	Synchronization period support: `0b0` TRCSYNCPR is read/write.
[24]	TRCERR	Indicates whether TRCVICTLR.TRCERR is implemented: `0b1` TRCERR is implemented.
[23:20]	EXLEVEL_NS	Exception levels implemented in Non-secure state. One bit for each exception level 0-3. `0b0000` No Non-secure exception levels are implemented.
[19:16]	EXLEVEL_S	Exception levels implemented in Secure state. One bit for each exception level 0-3. `0b1001` Secure exception levels EL0 and EL3 are implemented.
[15:12]	-	Reserved. RAZ/WI.
[11:0]	CCITMIN	Instruction trace cycle counting minimum threshold: `0x4` Minimum threshold is 4.

Related reference

11.7.1 Cortex®-R8 processor ETM register summary on page 11-254

Implementation-specific and identification registers on page 11-260

11.8.34 ID Register 4

The TRCIDR4 indicates the resources available in the Cortex-R8 processor ETM.

Usage constraints

There are no usage constraints.

Configurations

Available in all configurations.

Attributes

Register number: 124

Base offset `0x1F0`

Name: TRCIDR4

Type: RO

Reset: `0x01270124`

The following figure shows the TRCIDR4 bit assignments.

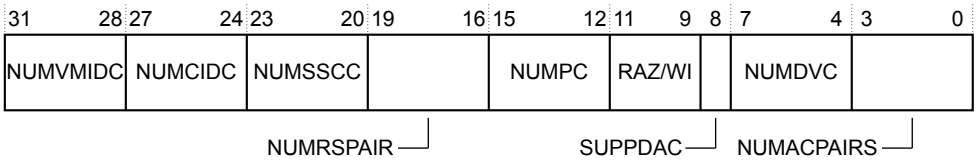


Figure 11-43 TRCIDR4 bit assignments

The following table shows the TRCIDR4 bit assignments.

Table 11-55 TRCIDR4 bit assignments

Bits	Name	Function
[31:28]	NUMVMIDC	Number of *Virtual Machine ID* (VMID) comparators implemented: `0b0000` VMID comparators are not implemented.
[27:24]	NUMCIDC	Number of Context ID comparators implemented: `0b0001` One context ID comparator is implemented.
[23:20]	NUMSSCC	Number of Single-Shot comparator controls implemented: `0b0010` Two single-shot comparator controls are implemented.
[19:16]	NUMRSPAIR	Number of resource selection pairs implemented: `0b0111` Eight resource selection pairs are implemented. The first is not counted.
[15:12]	NUMPC	Number of Cortex-R8 core comparator inputs implemented: `0b0000` Core comparator inputs are not implemented.
[11:9]	-	Reserved. RAZ/WI.

Table 11-55 TRCIDR4 bit assignments (continued)

Bits	Name	Function
[8]	SUPPDAC	Data address comparisons implemented: `0b1` Data address comparisons are supported.
[7:4]	NUMDVC	Number of data value comparators implemented: `0b0010` Two data value comparators are implemented.
[3:0]	NUMACPAIRS	Number of address comparator pairs implemented: `0b0100` Four address comparator pairs are implemented.

Related reference

11.7.1 Cortex®-R8 processor ETM register summary on page 11-254

Implementation-specific and identification registers on page 11-260

11.8.35 ID Register 5

The TRCIDR5 indicates the resources available in the Cortex-R8 processor ETM.

Usage constraints

There are no usage constraints.

Configurations

Available in all configurations.

Attributes

Register number: 125

Base offset `0x1F4`

Name: TRCIDR5

Type: RO

Reset: `0x28C70840`

The following figure shows the TRCIDR5 bit assignments.

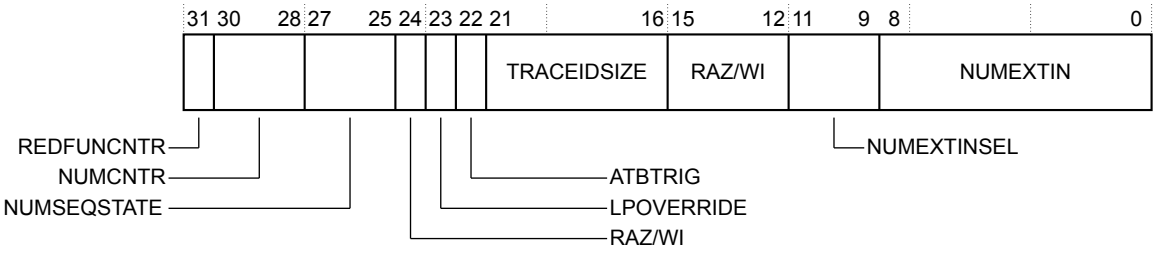


Figure 11-44 TRCIDR5 bit assignments

The following table shows the TRCIDR5 bit assignments.

Table 11-56 TRCIDR5 bit assignments

Bits	Name	Function
[31]	REDFUNCNTR	Reduced Function Counter implemented: `0b0` Reduced Function Counter not implemented
[30:28]	NUMCNTR	Number of counters implemented: `0b010` Two counters implemented
[27:25]	NUMSEQSTATE	Number of sequencer states implemented: `0b100` Four sequencer states implemented
[24]	-	Reserved. RAZ/WI
[23]	LPOVERRIDE	Low-power state override support: `0b1` Low-power state override support implemented
[22]	ATBTRIG	ATB trigger support: `0b1` ATB trigger support implemented
[21:16]	TRACEIDSIZE	Number of bits of trace ID: `0x07` Seven-bit trace ID implemented
[15:12]	-	Reserved. RAZ/WI.
[11:9]	NUMEXTINSEL	Number of external input selectors implemented: `0b100` Four external input selectors implemented
[8:0]	NUMEXTIN	Number of external inputs implemented: `0x40` 64 external inputs implemented

Related reference
11.7.1 Cortex®-R8 processor ETM register summary on page 11-254
Implementation-specific and identification registers on page 11-260

11.8.36 Resource Selection Registers 2-16

The TRCRSCTLRn control the trace resources.

Usage constraints
There are no usage constraints.

Configurations
Available in all configurations.

Attributes

Register number: 130-140

Base offset `0x208-0x240`

Name: TRCRSCTLRn

Type: RW

Reset: -

The following figure shows the TRCRSCTLRn bit assignments.

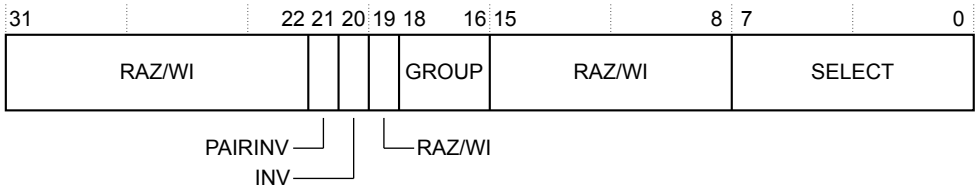


Figure 11-45 TRCRSCTLRn bit assignments

The following table shows the TRCRSCTLRn bit assignments.

Table 11-57 TRCRSCTLRn bit assignments

Bits	Name	Function
[31:22]	-	Reserved. RAZ/WI.
[21]	PAIRINV	Inverts the result of a combined pair of resources. This bit is only implemented on the lower register for a pair of resource selectors.
[20]	INV	Inverts the selected resources: `0b0` Resource is not inverted. `0b1` Resource is inverted.
[19]	-	Reserved. RAZ/WI.
[18:16]	GROUP	Selects a group of resources.
[15:8]	-	Reserved. RAZ/WI.
[7:0]	SELECT	Selects one or more resources from the desired group. One bit is provided per resource from the group.

Related reference

11.7.1 Cortex®-R8 processor ETM register summary on page 11-254

Resource selection registers on page 11-260

11.8.37 Single-Shot Comparator Control Registers 0-1

The TRCSSCCRn control the single-shot comparators.

Usage constraints

There are no usage constraints.

Configurations

Available in all configurations.

Attributes

Register number: 160-161

Base offset `0x280-0x284`

Name: TRCSSCCRn

Type: RW

Reset: -

The following figure shows the TRCSSCCRn bit assignments.

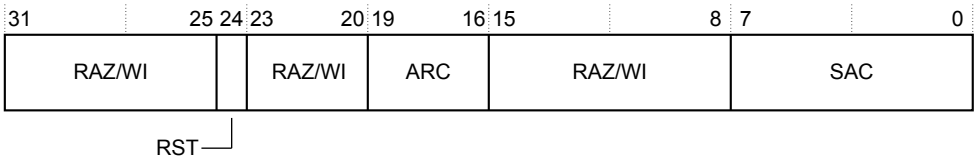


Figure 11-46 TRCSSCCRn bit assignments

The following table shows the TRCSSCCRn bit assignments.

Table 11-58 TRCSSCCRn bit assignments

Bits	Name	Function
[31:25]	-	Reserved. RAZ/WI.
[24]	RST	Enables the single-shot comparator resource to be reset when it occurs, to enable another comparator match to be detected: `0b1` Reset enabled. Multiple matches can occur.
[23:20]	-	Reserved. RAZ/WI.
[19:16]	ARC	Selects one or more address range comparators for single-shot control. One bit is provided for each implemented address range comparator.
[15:8]	-	Reserved. RAZ/WI.
[7:0]	SAC	Selects one or more single address comparators for single-shot control. One bit is provided for each implemented single address comparator.

Related reference

11.7.1 Cortex®-R8 processor ETM register summary on page 11-254

Single-shot comparator registers on page 11-261

11.8.38 Single-Shot Comparator Status Registers 0-1

The TRCSSCSRn indicate the status of the single-shot comparators. TRCSSCSR0 is sensitive to instruction addresses. TRCSSCSR1 is sensitive to data addresses and values.

Usage constraints

There are no usage constraints.

Configurations

Available in all configurations.

Attributes

Register number: 168-169

Base offset `0x2A0-0x2A4`

Name: TRCSSCSRn

Type: RW

Reset: -

The following figure shows the TRCSSCSRn bit assignments.

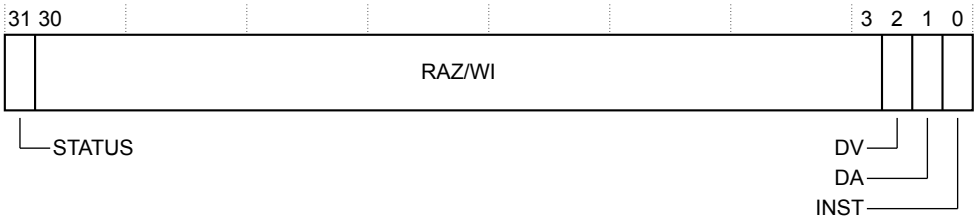


Figure 11-47 TRCSSCSRn bit assignments

The following table shows the TRCSSCSR0 bit assignments.

Table 11-59 TRCSSCSR0 bit assignments

Bits	Name	Function
[31]	STATUS	Single-shot status. This indicates whether any of the selected comparators have matched: `0b0` Match has not occurred. `0b1` Match has occurred at least once. When programming the Cortex-R8 processor ETM, if TRCSSCCRn.RST is `0b0`, the STATUS bit must be explicitly written to `0b0` to enable this single-shot comparator control.
[30:3]	-	Reserved. RAZ/WI.
[2]	DV	Data value comparator support: `0b0` Single-shot data value comparisons not supported.
[1]	DA	Data address comparator support: `0b0` Single-shot data address comparisons not supported.
[0]	INST	Instruction address comparator support: `0b1` Single-shot instruction address comparisons supported.

The following table shows the TRCSSCSR1 bit assignments.

Table 11-60 TRCSSCSR1 bit assignments

Bits	Name	Function
[31]	STATUS	Single-shot status. This indicates whether any of the selected comparators have matched: `0b0` Match has not occurred. `0b1` Match has occurred at least once. When programming the Cortex-R8 processor ETM, this bit must be explicitly written to `0b0` to enable this single-shot comparator control.
[30:3]	-	Reserved. RAZ/WI.
[2]	DV	Data value comparator support: `0b1` Single-shot data value comparisons supported.

Table 11-60 TRCSSCSR1 bit assignments (continued)

Bits	Name	Function
[1]	DA	Data address comparator support: `0b1` Single-shot data address comparisons supported.
[0]	INST	Instruction address comparator support: `0b0` Single-shot instruction address comparisons not supported.

Related reference

11.7.1 Cortex®-R8 processor ETM register summary on page 11-254

Single-shot comparator registers on page 11-261

11.8.39 OS Lock Access Register

The TRCOSLAR sets and clears the OS Lock, to lock out external debugger accesses to the Cortex-R8 processor ETM registers.

Usage constraints

There are no usage constraints.

Configurations

Available in all configurations.

Attributes

Register number: 192

Base offset `0x300`

Name: TRCOSLAR

Type: WO

Reset: -

The following figure shows the TRCOSLAR bit assignments.

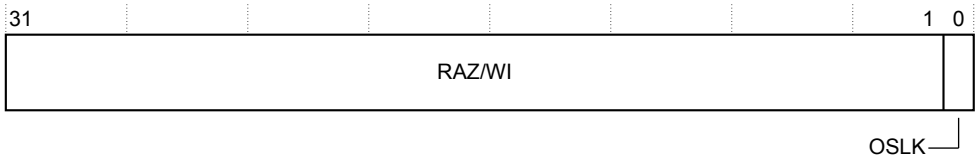


Figure 11-48 TRCOSLAR bit assignments

The following table shows the TRCOSLAR bit assignments.

Table 11-61 TRCOSLAR bit assignments

Bits	Name	Function
[31:1]	-	Reserved. RAZ/WI.
[0]	OSLK	OS Lock key value: `0b0` Unlock the OS Lock. `0b1` Lock the OS Lock.

Related reference

11.7.1 Cortex®-R8 processor ETM register summary on page 11-254

OS lock and power control registers on page 11-262

11.8.40 OS Lock Status Register

The TRCOSLSR returns the status of the OS Lock.

Usage constraints

There are no usage constraints.

Configurations

Available in all configurations.

Attributes

Register number: 193

Base offset `0x304`

Name: TRCOSLSR

Type: RO

Reset: -

The following figure shows the TRCOSLSR bit assignments.

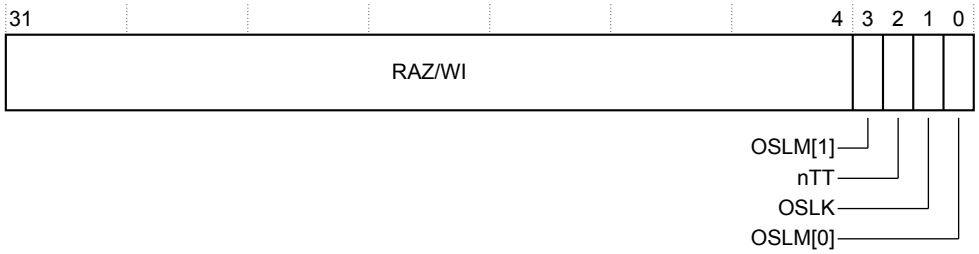


Figure 11-49 TRCOSLSR bit assignments

The following figure shows the TRCOSLSR bit assignments.

Table 11-62 TRCOSLSR bit assignments

Bits	Name	Function
[31:4]	-	Reserved. RAZ/WI.
[3]	OSLM[1]	OS Lock model[1]. This bit is combined with OSLM[0] to form a two-bit field that indicates the OS Lock model is implemented. The value of this field is always `0b10`, indicating that the OS Lock is implemented.
[2]	nTT	This bit is RAZ, which indicates that software must perform a 32-bit write to update the TRCOSLAR.
[1]	OSLK	OS Lock status bit: `0b0` OS Lock is unlocked. `0b1` OS Lock is locked.
[0]	OSLM[0]	OS Lock model[0]. This bit is combined with OSLM[1] to form a two-bit field that indicates the OS Lock model is implemented. The value of this field is always `0b10`, indicating that the OS Lock is implemented.

Related reference

11.7.1 Cortex®-R8 processor ETM register summary on page 11-254

OS lock and power control registers on page 11-262

11.8.41 Power Down Control Register

The TRCPDCR requests the system power controller to keep the Cortex-R8 processor ETM powered up.

Usage constraints

There are no usage constraints.

Configurations

Available in all configurations.

Attributes

Register number: 196

Base offset `0x310`

Name: TRCPDCR

Type: RW

Reset: `0x00000000`

The following figure shows the TRCPDCR bit assignments.

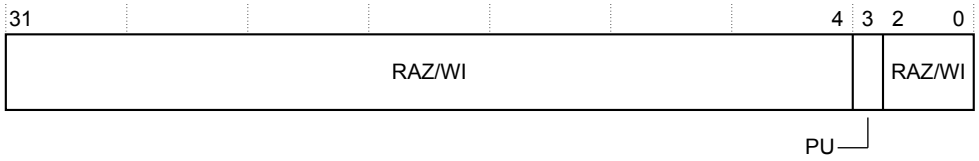


Figure 11-50 TRCPDCR bit assignments

The following table shows the TRCPDCR bit assignments.

Table 11-63 TRCPDCR bit assignments

Bits	Name	Function
[31:4]	-	Reserved. RAZ/WI.
[3]	PU	Power up request, to request that power to the Cortex-R8 processor ETM and access to the trace registers is maintained: `0b0` Power not requested. `0b1` Power requested. This bit is reset to `0b0` on a trace unit reset.
[2:0]	-	Reserved. RAZ/WI.

Related reference

11.7.1 Cortex®-R8 processor ETM register summary on page 11-254

OS lock and power control registers on page 11-262

11.8.42 Power Down Status Register

The TRCPDSR indicates the powerdown status of the Cortex-R8 processor ETM.

Usage constraints

There are no usage constraints.

Configurations

Available in all configurations.

Attributes

Register number: 197

Base offset `0x314`

Name: TRCPDSR

Type: RO

Reset: `0x00000023`

The following figure shows the TRCPDSR bit assignments.

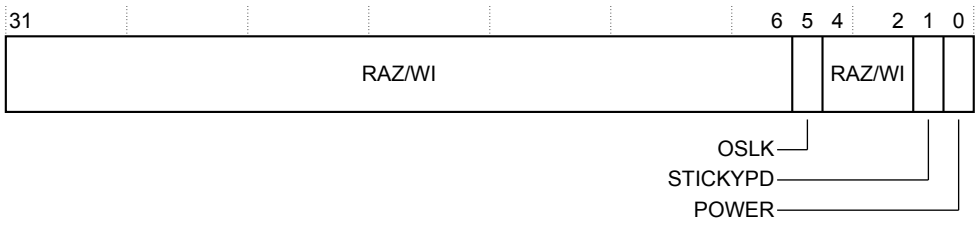


Figure 11-51 TRCPDSR bit assignments

The following table shows the TRCPDSR bit assignments.

Table 11-64 TRCPDSR bit assignments

Bits	Name	Function
[31:6]	-	Reserved. RAZ/WI.
[5]	OSLK	OS lock status.
[4:2]	-	Reserved. RAZ/WI.
[1]	STICKYPD	Sticky powerdown state. `0b0` Trace register power has not been removed since the TRCPDSR was last read. `0b1` Trace register power has been removed since the TRCPDSR was last read. This bit is set to `0b1` when power to the Cortex-R8 processor ETM registers is removed, to indicate that programming state has been lost. It is cleared after a read of the TRCPDSR.
[0]	POWER	Indicates the Cortex-R8 processor ETM is powered: `0b1` Cortex-R8 processor ETM is powered. All registers are accessible. If a system implementation allows the ETM to be powered off independently of the debug power domain, the system must handle accesses to the ETM appropriately.

Related reference

11.7.1 Cortex®-R8 processor ETM register summary on page 11-254

OS lock and power control registers on page 11-262

11.8.43 Address Comparator Value Registers 0-7

The TRCACVRn indicates the address for the data address comparators.

Usage constraints

There are no usage constraints.

Configurations

Available in all configurations.

Attributes

Register number: 256-271

Base offset `0x400-0x43C`

Name: TRCACVRn

Type: RW

Reset: -

The following figure shows the TRCACVRn bit assignments.

Figure 11-52 TRCACVRn bit assignments

The following table shows the TRCACVRn bit assignments.

Table 11-65 TRCACVRn bit assignments

Bits	Name	Function
[31:0]	ADDRESS	The address value to compare against

Related reference

11.7.1 Cortex®-R8 processor ETM register summary on page 11-254

Comparator registers on page 11-263

11.8.44 Address Comparator Access Type Registers 0-7

The TRCACATRn controls the access for the data address comparators.

Usage constraints

There are no usage constraints.

Configurations

Available in all configurations.

Attributes

Register number: 288-303

Base offset `0x480-0x4BC`

Name: TRCACATRn

Type: RW

Reset: `0x480-0x4BC`

The following figure shows the TRCACATR0 bit assignments.

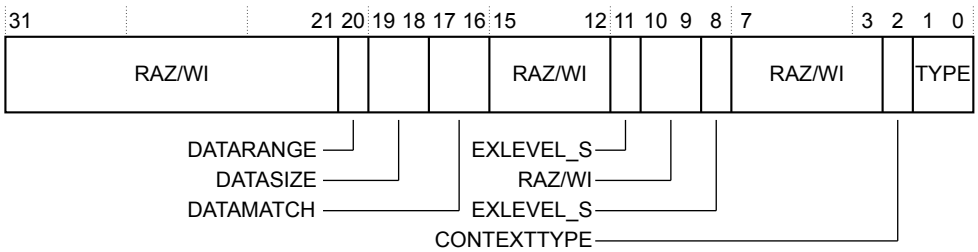


Figure 11-53 TRCACATR0 bit assignments

The following table shows the TRCACATR0 bit assignments.

Table 11-66 TRCACATR0 bit assignments

Bits	Name	Function
[31:21]	-	Reserved. RAZ/WI.
[20]	DATARANGE	Data value comparison range control, to select whether the data value comparison is made against the single address comparator or the address range comparator: `0b0` Only the single address comparator matches. `0b1` Only the address range comparator matches.
[19:18]	DATASIZE	Data value comparison size control: `0b00` Byte size. `0b01` Halfword size. `0b10` Word size. `0b11` Doubleword size.
[17:16]	DATAMATCH	Data value comparison control: `0b01` Comparator matches only if the data value comparison matches. `0b11` Comparator matches only if the data value comparison does not match.
[15:12]	-	Reserved. RAZ/WI.
[11]	EXLEVEL_S	Indicates whether the comparator matches in exception level 3 in Secure state: `0b1` The comparator must not match in this exception level.
[10:9]	-	Reserved. RAZ/WI.
[8]	EXLEVEL_S	Indicates whether the comparator matches in exception level 0 in Secure state: `0b1` The comparator must not match in this exception level.
[7:3]	-	Reserved. RAZ/WI.
[2]	CONTEXTTYPE	Indicates whether the context comparator is used in the comparison: `0b0` Use no context comparators. `0b1` Use the Context ID comparator.
[1:0]	TYPE	The type of comparison: `0b00` Instruction address. `0b01` Data load address. `0b10` Data store address. `0b11` Data load or store address.

Note

TRCACATR2 is functionally identical to TRCACATR0.

The figure shows the TRCACATR1 bit assignments.

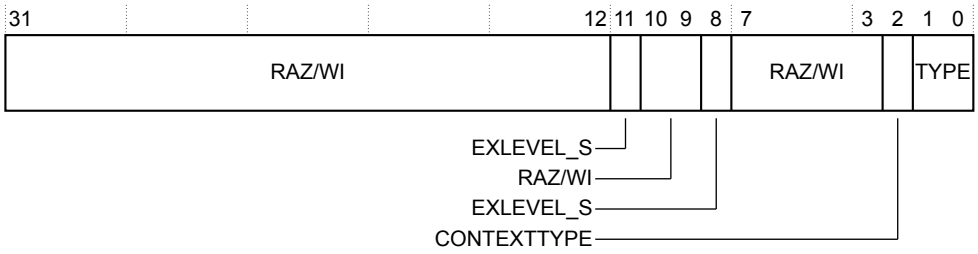


Figure 11-54 TRCACATR1 bit assignments

The table shows the TRCACATR1 bit assignments.

Table 11-67 TRCACATR1 bit assignments

Bits	Name	Function
[31:12]	-	Reserved. RAZ/WI.
[11]	EXLEVEL_S	Indicates whether the comparator matches in exception level 3 in Secure state: `0b1` The comparator must not match in this exception level.
[10:9]	-	Reserved. RAZ/WI.
[8]	EXLEVEL_S	Indicates whether the comparator matches in exception level 0 in Secure state: `0b1` The comparator must not match in this exception level.
[7:3]	-	Reserved. RAZ/WI.
[2]	CONTEXTTYPE	Indicates whether the context comparator is used in the comparison: `0b0` Use no context comparators. `0b1` Use the Context ID comparator.
[1:0]	TYPE	The type of comparison: `0b00` Instruction address. `0b01` Data load address. `0b10` Data store address. `0b11` Data load or store address.

Note

TRCACATR3-7 are functionally identical to TRCACATR1.

Related reference

11.7.1 Cortex®-R8 processor ETM register summary on page 11-254

Comparator registers on page 11-263

11.8.45 Data Value Comparator Value Registers 0-1

The TRCDVCVRn indicates the value for the data value comparators.

Usage constraints

There are no usage constraints.

Configurations

Available in all configurations.

Attributes

Register number: 320-321

Base offset `0x500-0x504`

Name: TRCDVCVRn

Type: RW

Reset: -

The following figure shows the TRCDVCVRn bit assignments.

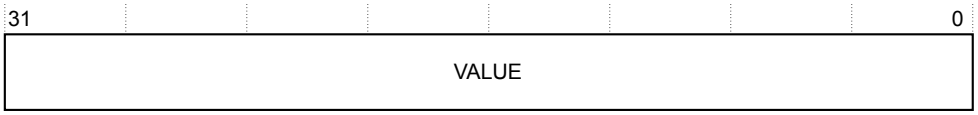


Figure 11-55 TRCDVCVRn bit assignments

The following table shows the TRCDVCVRn bit assignments.

Table 11-68 TRCDVCVRn bit assignments

Bits	Name	Function
[31:0]	VALUE	The data value to compare against

Related reference

11.7.1 Cortex®-R8 processor ETM register summary on page 11-254

Comparator registers on page 11-263

11.8.46 Data Value Comparator Mask Registers 0-1

The TRCDVCMRn control the mask value for the data value comparators.

Usage constraints

There are no usage constraints.

Configurations

Available in all configurations.

Attributes

Register number: 352-359

Base offset `0x580-0x59C`

Name: TRCDVCMRn

Type: RW

Reset: -

The following figure shows the TRCDVCMRn bit assignments.

Figure 11-56 TRCDVCMRn bit assignments

The following table shows the TRCDVCMRn bit assignments.

Table 11-69 TRCDVCMRn bit assignments

Bits	Name	Function
[31:0]	MASK	The mask value to apply to the data value comparison

Related reference

11.7.1 Cortex®-R8 processor ETM register summary on page 11-254

Comparator registers on page 11-263

11.8.47 Context ID Comparator Control Register 0

The TRCCIDCCTLR0 controls the mask value for the context ID comparators.

Usage constraints

There are no usage constraints.

Configurations

Available in all configurations.

Attributes

Register number: 416

Base offset `0x680`

Name: TRCCIDCCTLRn

Type: RW

Reset: -

The following figure shows the TRCCIDCCTLR0 bit assignments.

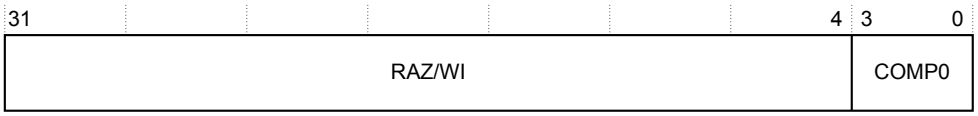


Figure 11-57 TRCCIDCCTLR0 bit assignments

The following table shows the TRCCIDCCTLR0 bit assignments.

Table 11-70 TRCCIDCCTLR0 bit assignments

Bits	Name	Function
[31:4]	-	Reserved. RAZ/WI
[3:0]	COMP0	The mask value to apply to the Context ID comparator

Related reference

11.7.1 Cortex®-R8 processor ETM register summary on page 11-254

Comparator registers on page 11-263

11.8.48 Context ID Comparator Value Register 0

The TRCCIDCVR0 indicates the value for the context ID comparators.

Usage constraints

There are no usage constraints.

Configurations

Available in all configurations.

Attributes

Register number: 384

Base offset `0x600`

Name: TRCCIDCVR0

Type: RW

Reset: -

The following figure shows the TRCCIDCVR0 bit assignments.

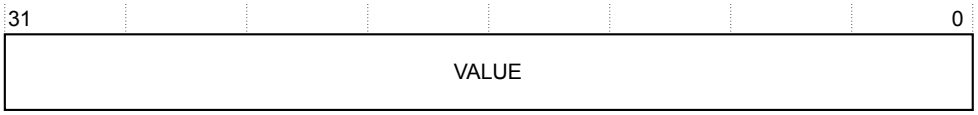


Figure 11-58 TRCCIDCVR0 bit assignments

The following table shows the TRCCIDCVR0 bit assignments.

Table 11-71 TRCCIDCVR0 bit assignments

Bits	Value	Function
[31:0]	VALUE	The Context ID value to compare with the current Context ID

Related reference

11.7.1 Cortex®-R8 processor ETM register summary on page 11-254

Comparator registers on page 11-263

11.8.49 Integration Mode Control Register

The TRCITCTRL enables topology detection or integration testing by putting the Cortex-R8 processor ETM into integration mode.

Usage constraints

Arm recommends that you perform a debug reset after using integration mode.

Configurations

Available in all configurations.

Attributes

Register number: 960

Base offset `0xF00`

Name: TRCITCTRL

Type: RW

Reset: `0x00000000`

The following figure shows the TRCITCTRL bit assignments.

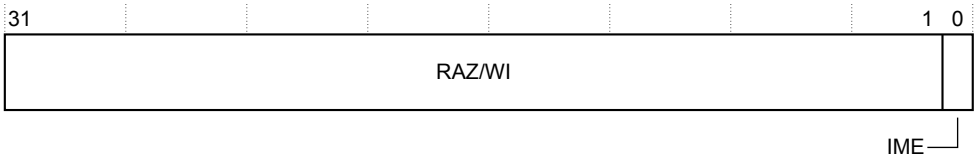


Figure 11-59 TRCITCTRL bit assignments

The following table shows the TRCITCTRL bit assignments.

Table 11-72 TRCITCTRL bit assignments

Bits	Name	Function
[31:1]	-	Reserved. RAZ/WI.
[0]	IME	Integration mode enable: `0b0` Cortex-R8 processor ETM is not in integration mode. This is the reset value. `0b1` Cortex-R8 processor ETM is in integration mode.

Related reference

11.7.1 Cortex®-R8 processor ETM register summary on page 11-254

CoreSight™ management registers on page 11-265

11.8.50 Claim Tag Set Register

The TRCCLAIMSET sets bits in the claim tag and determines the number of claim tag bits implemented.

Usage constraints

There are no usage constraints.

Configurations

Available in all configurations.

Attributes

Register number: 1000

Base offset `0xFA0`

Name: TRCCLAIMSET

Type: RW

Reset: `0x00000000`

The following figure shows the TRCCLAIMSET bit assignments.

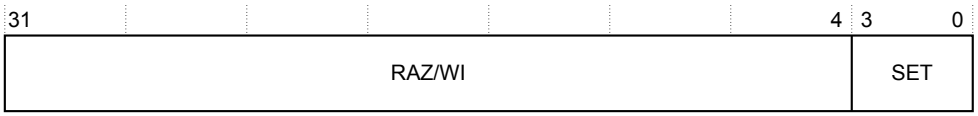


Figure 11-60 TRCCLAIMSET bit assignments

The following table shows the TRCCLAIMSET bit assignments.

Table 11-73 TRCCLAIMSET bit assignments

Bits	Name	Function
[31:4]	-	Reserved. RAZ/WI.
[3:0]	SET	On reads, for each bit: `0b0` Claim tag bit is not implemented. `0b1` Claim tag bit is implemented. On writes, for each bit: `0b0` Has no effect. `0b1` Sets the relevant bit of the claim tag.

Related reference

11.7.1 Cortex®-R8 processor ETM register summary on page 11-254

CoreSight™ management registers on page 11-265

11.8.51 Claim Tag Clear Register

The TRCCLAIMCLR clears bits in the claim tag and determines the current value of the claim tag.

Usage constraints

There are no usage constraints.

Configurations

Available in all configurations.

Attributes

Register number: 1001

Base offset `0xFA4`

Name: TRCCLAIMCLR

Type: RW

Reset: `0x00000000`

The following figure shows the TRCCLAIMCLR bit assignments.

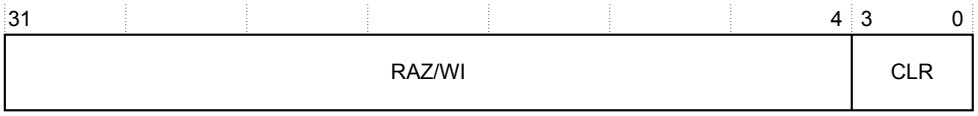


Figure 11-61 TRCCLAIMCLR bit assignments

The following table shows the TRCCLAIMCLR bit assignments.

Table 11-74 TRCCLAIMCLR bit assignments

Bits	Name	Function
[31:4]	-	Reserved. RAZ/WI.
[3:0]	CLR	On reads, for each bit: `0b0` Claim tag bit is not set. `0b1` Claim tag bit is set. On writes, for each bit: `0b0` Has no effect. `0b1` Clears the relevant bit of the claim tag.

Related reference

11.7.1 Cortex®-R8 processor ETM register summary on page 11-254

CoreSight™ management registers on page 11-265

11.8.52 Device Affinity Register

The TRCDEVAFF0 enables the Cortex-R8 processor ETM to determine which core in the Cortex-R8 processor the component relates to.

Usage constraints

There are no usage constraints.

Configurations

Available in all configurations.

Attributes

Register number: 1002

Base offset `0xFA8`

Name: TRCDEVAFF0

Type: RO

Reset: -

The following figure shows the TRCDEVAFF0 bit assignments.

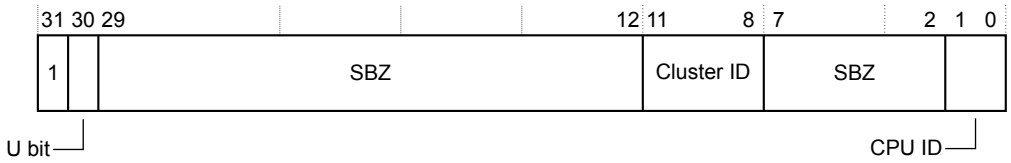


Figure 11-62 TRCDEVAFF0 bit assignments

The following table shows the TRCDEVAFF0 bit assignments.

Table 11-75 TRCDEVAFF0 bit assignments

Bits	Name	Function
[31]	-	Indicates the register uses the new multiprocessor format. This is always `0b1`.
[30]	U bit	Multiprocessing Extensions: `0b0` Indicates the Cortex-R8 processor is a multiprocessor configuration.
[29:12]	-	Reserved. SBZ.
[11:8]	Cluster ID	Value read in **CLUSTERID** configuration pins. It identifies a Cortex-R8 processor cluster in a system that has several Cortex-R8 processor clusters present.
[7:2]	-	Reserved. SBZ.
[1:0]	CPU ID	Indicates the core number in the multiprocessor configuration: `0x00` Core 0. `0x01` Core 1. `0x10` Core 2. `0x11` Core 3.

Related reference

11.7.1 Cortex®-R8 processor ETM register summary on page 11-254

CoreSight™ management registers on page 11-265

11.8.53 Software Lock Access Register

The TRCLAR controls access to registers using the memory-mapped interface.

When the software lock is set, write accesses using the memory-mapped interface to all Cortex-R8 processor ETM registers are ignored except for write accesses to the TRCLAR.

When the software lock is set, read accesses of TRCPDSR do not change the TRCPDSR.STICKYPD bit. Read accesses of all other registers are not affected.

Usage constraints

There are no usage constraints.

Configurations

Available in all configurations.

Attributes

Register number: 1004

Base offset `0xFB0`

Name: TRCLAR

Type: WO

Reset: -

The following figure shows the TRCLAR bit assignments.

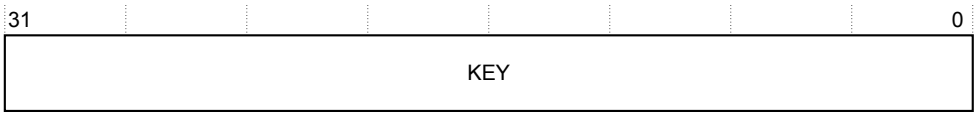


Figure 11-63 TRCLAR bit assignments

The following table shows the TRCLAR bit assignments.

Table 11-76 TRCLAR bit assignments

Bits	Name	Function
[31:0]	KEY	Software lock key value: `0xC5ACCE55` Clear the software lock. All other write values set the software lock.

Related reference

11.7.1 Cortex®-R8 processor ETM register summary on page 11-254

CoreSight™ management registers on page 11-265

11.8.54 Software Lock Status Register

The TRCLSR determines if the software lock is implemented and indicates the current status of the software lock.

Usage constraints

There are no usage constraints.

Configurations

Available in all configurations.

Attributes

Register number: 1005

Base offset `0xFB4`

Name: TRCLSR

Type: RO

Reset: -

The following figure shows the TRCLSR bit assignments.

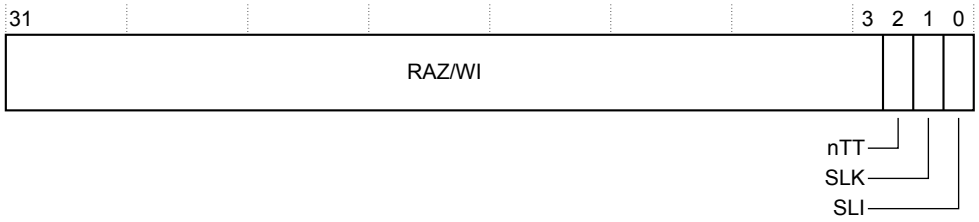


Figure 11-64 TRCLSR bit assignments

The following table shows the TRCLSR bit assignments.

Table 11-77 TRCLSR bit assignments

Bits	Name	Function
[31:3]	-	Reserved. RAZ/WI.
[2]	nTT	Indicates size of TRCLAR: `0b0` TRCLAR is always 32 bits.
[1]	SLK	Software lock status: `0b0` Software lock is clear. `0b1` Software lock is set.
[0]	SLI	Indicates whether the software lock is implemented on this interface. `0b1` Software lock is implemented on this interface.

Related reference
11.7.1 Cortex®-R8 processor ETM register summary on page 11-254
CoreSight™ management registers on page 11-265

11.8.55 Authentication Status Register

The TRCAUTHSTATUS indicates the current level of tracing permitted by the system.

Usage constraints
There are no usage constraints.

Configurations
Available in all configurations.

Attributes

Register number: 1006

Base offset `0xFB8`

Name: TRCAUTHSTATUS

Type: RO

Reset: -

The following figure shows the TRCAUTHSTATUS bit assignments.

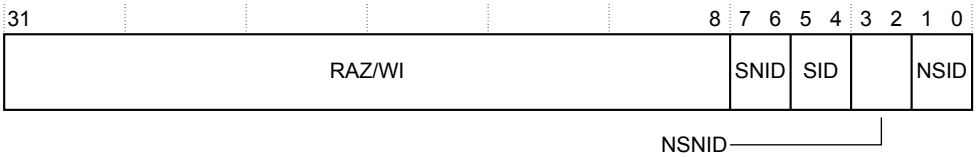


Figure 11-65 TRCAUTHSTATUS bit assignments

The following table shows the TRCAUTHSTATUS bit assignments.

Table 11-78 TRCAUTHSTATUS bit assignments

Bits	Name	Function
[31:8]	-	Reserved. RAZ/WI.
[7:6]	SNID	Secure Non-Invasive Debug: `0b10` Secure Non-Invasive Debug implemented but disabled. `0b11` Secure Non-Invasive Debug implemented and enabled.
[5:4]	SID	Secure Invasive Debug: `0b10` Secure Invasive Debug implemented but disabled. `0b11` Secure Invasive Debug implemented and enabled.
[3:2]	NSNID	Non-secure Non-Invasive Debug: `0b00` Non-secure Non-Invasive Debug not implemented.
[1:0]	NSID	Non-secure Invasive Debug: `0b00` Non-secure Invasive Debug not implemented.

Related reference
11.7.1 Cortex®-R8 processor ETM register summary on page 11-254
CoreSight™ management registers on page 11-265

11.8.56 Device Architecture Register

The TRCDEVARCH identifies the Cortex-R8 processor ETM as an ETMv4 component.

Usage constraints
There are no usage constraints.

Configurations
Available in all configurations.

Attributes

Register number: 1007

Base offset `0xFBC`

Name: TRCDEVARCH

Type: RO

Reset: `0x47704A17`

The following figure shows the TRCDEVARCH bit assignments.

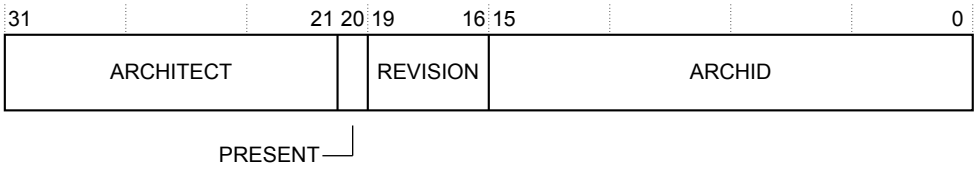


Figure 11-66 TRCDEVARCH bit assignments

The following table shows the TRCDEVARCH bit assignments.

Table 11-79 TRCDEVARCH bit assignments

Bits	Name	Function
[31:21]	ARCHITECT	Defines the architect of the component: `0x43B` Arm
[20]	PRESENT	Indicates the presence of this register: `0b1` Register is present
[19:16]	REVISION	Architecture revision: `0b0000` Architecture revision 0
[15:0]	ARCHID	Architecture ID: `0x4A13` ETMv4 component

Related reference

11.7.1 Cortex®-R8 processor ETM register summary on page 11-254

CoreSight™ management registers on page 11-265

11.8.57 Device ID Register

The TRCDEVID indicates the capabilities of the Cortex-R8 processor ETM.

Usage constraints

There are no usage constraints.

Configurations

Available in all configurations.

Attributes

Register number: 1010

Base offset `0xFC8`

Name: TRCDEVID

Type: RO

Reset: `0x00000000`

The following figure shows the TRCDEVID bit assignments.

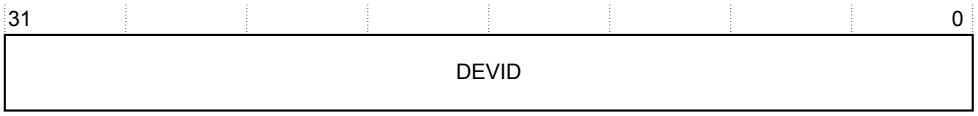


Figure 11-67 TRCDEVID bit assignments

The following table shows the TRCDEVID bit assignments.

Table 11-80 TRCDEVID bit assignments

Bits	Name	Function
[31:0]	DEVID	RAZ. There are no component-defined capabilities.

Related reference
11.7.1 Cortex®-R8 processor ETM register summary on page 11-254
CoreSight™ management registers on page 11-265

11.8.58 Device Type Register

The TRCDEVTYPE indicates the type of the component.

Usage constraints
There are no usage constraints.

Configurations
Available in all configurations.

Attributes

Register number: 1011

Base offset `0xFCC`

Name: TRCDEVTYPE

Type: RO

Reset: `0x00000013`

The following figure shows the TRCDEVTYPE bit assignments.

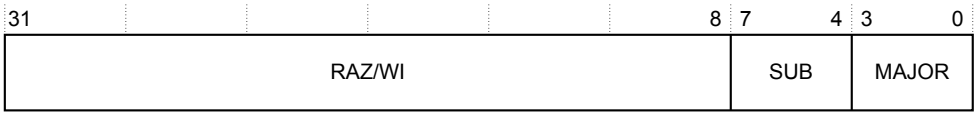


Figure 11-68 TRCDEVTYPE bit assignments

The following table shows the TRCDEVTYPE bit assignments.

Table 11-81 TRCDEVTYPE bit assignments

Bits	Name	Function
[31:8]	-	Reserved. RAZ/WI
[7:4]	SUB	The subtype of the component: `0b0001` Processor trace
[3:0]	MAJOR	The main type of the component: `0b0011` Trace source

Related reference
11.7.1 Cortex®-R8 processor ETM register summary on page 11-254
CoreSight™ management registers on page 11-265

11.8.59 Peripheral Identification Registers

The TRCPIDR0-7 provide the standard Peripheral ID required by all CoreSight components.

See the *Arm® Embedded Trace Macrocell Architecture Specification ETMv4* for more information.

Usage constraints

Only bits[7:0] of each register are used. This means that TRCPIDR0-7 define a single 64-bit *Peripheral ID*, as the figure shows.

Configurations

Available in all configurations.

Attributes

Register number: 1012-1019

Base offset `0xFD0-0xFEC`

Name: TRCPIDRn

Type: RO

Reset: -

The following figure shows the mapping between TRCPIDR0-7 and the single 64-bit *Peripheral ID* value.

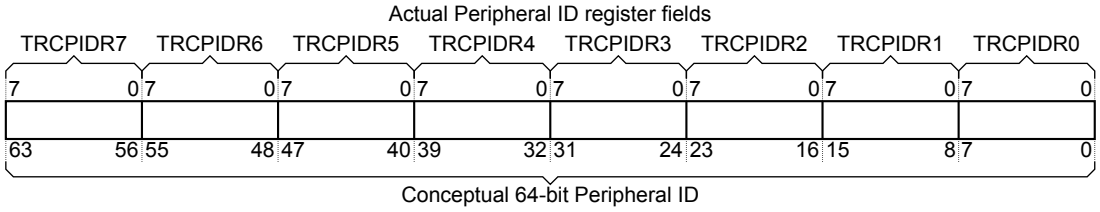
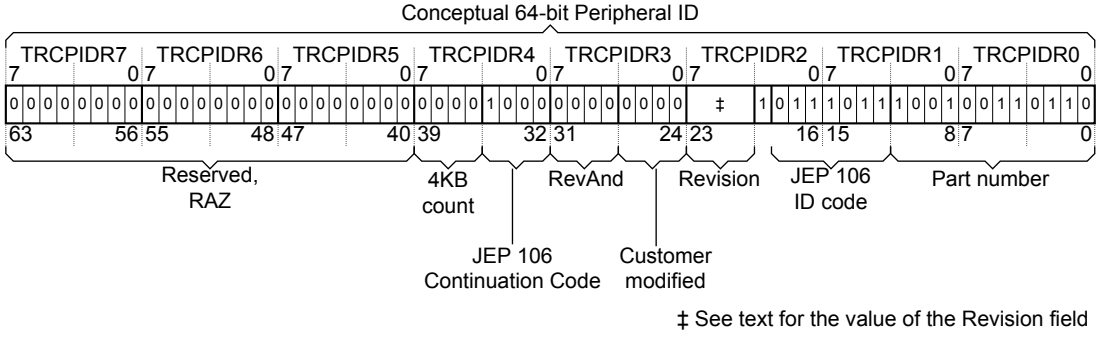


Figure 11-69 Mapping between TRCPIDR0-7 and the Peripheral ID value

The following figure shows the Peripheral ID bit assignments in the single conceptual Peripheral ID register.



‡ See text for the value of the Revision field

Figure 11-70 Peripheral ID fields

The following table shows the values of the fields when reading this set of registers. The *Arm® Embedded Trace Macrocell Architecture Specification ETMv4* gives more information about many of these fields.

Table 11-82 TCRPIDR0-7 bit assignments

Register	Register number	Register offset	Bits	Value	Description
TRCPIDR7	0x3F7	0xFDC	[31:8]	-	Unused, read UNDEFINED.
			[7:0]	0x00	Reserved for future use, RAZ.
TRCPIDR6	0x3F6	0xFD8	[31:8]	-	Unused, read UNDEFINED.
			[7:0]	0x00	Reserved for future use, RAZ.
TRCPIDR5	0x3F5	0xFD4	[31:8]	-	Unused, read UNDEFINED.
			[7:0]	0x00	Reserved for future use, RAZ.
TRCPIDR4	0x3F4	0xFD0	[31:8]	-	Unused, read UNDEFINED.
			[7:4]	0x0	n, where 2^n is number of 4KB blocks used.
			[3:0]	0x4	JEP 106 continuation code.
TRCPIDR3	0x3FB	0xFEC	[31:8]	-	Unused, read UNDEFINED.
			[7:4]	0x0	RevAnd (at top level). Manufacturer revision number.
			[3:0]	0x0	Customer Modified. 0x0 indicates from Arm.
TRCPIDR2	0x3FA	0xFE8	[31:8]	-	Unused, read UNDEFINED.
			[7:4]	bp	Revision Number of Peripheral. This value is the same as the Implementation revision field of the TRCIDR, see *11.8.31 ID Register 1* on page 11-300.
			[3]	0b1	Always 1. Indicates that a JEDEC assigned value is used.
			[2:0]	0b011	JEP 106 identity code [6:4].
TRCPIDR1	0x3F9	0xFE4	[31:8]	-	Unused, read UNDEFINED.
			[7:4]	0b1011	JEP 106 identity code [3:0].
			[3:0]	0x9	Part Number[11:8]. Upper *Binary Coded Decimal* (BCD) value of Device Number.
TRCPIDR0	0x3F8	0xFE0	[31:8]	-	Unused, read UNDEFINED.
			[7:0]	0x37	Part Number [7:0]. Middle and Lower BCD value of Device Number.

Note

In the *TCRPIDR0-7 bit assignments* table, the *11.8.59 Peripheral Identification Registers* on page 11-328 are listed in order of register name, from most significant (TRCPIDR7) to least significant (TRCPIDR0). This does not match the order of the register offsets. Similarly, in the *TRCCIDR0-3 bit assignments* table

bp See the Description column for details.

the *11.8.60 Component Identification Registers* on page 11-331 are listed in order of register name, from most significant (TRCCIDR3) to least significant (TRCCIDR0).

Related reference

11.7.1 Cortex®-R8 processor ETM register summary on page 11-254

CoreSight™ management registers on page 11-265

11.8.31 ID Register 1 on page 11-300

11.8.59 Peripheral Identification Registers on page 11-328

11.8.60 Component Identification Registers on page 11-331

11.8.60 Component Identification Registers

The TRCCIDR0-3 identify the ETM as a CoreSight component.

For more information, see the *Arm® Embedded Trace Macrocell Architecture Specification ETMv4*.

Usage constraints

Only bits[7:0] of each register are used. This means that TRCCIDR0-3 define a single 32-bit Component ID, as the figure shows.

Configurations

Available in all configurations.

Attributes

Register number: 1020-1023

Base offset `xFF0-0xFFC`

Name: TRCCIDRn

Type: RO

Reset: -

The following figure shows the mapping between TRCCIDR0-3 and the single 32-bit *Component ID* value.

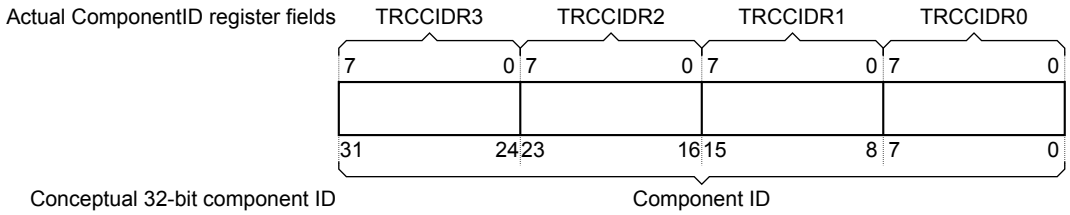


Figure 11-71 Mapping between TRCCIDR0-3 and the Component ID value

The following table shows the Component ID bit assignments in the single conceptual Component ID register.

Table 11-83 TRCCIDR0-3 bit assignments

Register	Register number	Register offset	Bits	Value	Description
TRCCIDR3	`0x3FF`	`0xFFC`	[31:8]	-	Unused, read UNDEFINED.
			[7:0]	`0xB1`	Component identifier, bits[31:24].
TRCCIDR2	`0x3FE`	`0xFF8`	[31:8]	-	Unused, read UNDEFINED.
			[7:0]	`0x05`	Component identifier, bits[23:16].
TRCCIDR1	`0x3FD`	`0xFF4`	[31:8]	-	Unused, read UNDEFINED.
			[7:4]	`0x9`	Component class (component identifier, bits[15:12]).
			[3:0]	`0x0`	Component identifier, bits[11:8].
TRCCIDR0	`0x3FC`	`0xFF0`	[31:8]	-	Unused, read UNDEFINED.
			[7:0]	`0x0D`	Component identifier, bits[7:0].

Related reference

11.7.1 Cortex®-R8 processor ETM register summary on page 11-254

CoreSight™ management registers on page 11-265

11.8.61 Integration Test Registers

To access the Integration Test Registers registers, you must first set bit[0] of the Integration Mode Control Register to `0b1`.

- You can use the write-only Integration Test Registers to set the outputs of some of the ETM signals. The following table shows the signals that can be controlled in this way.
- You can use the read-only Integration Test Registers to read the state of some of the ETM input signals. The *Input signals that the Integration Test Registers can read* table shows the signals that can be read in this way.

See the *Arm® Embedded Trace Macrocell Architecture Specification ETMv4* for details of TRCITCTRL.

Output signals that the Integration Test Registers can control

Details of the output signals that the Integration Test Registers can control.

Table 11-84 Output signals that the Integration Test Registers can control

Signal	Register	Bits	Register description
AFREADYMDx	TRCITDATBOUTR	[1]	See *Integration Data ATB Out Register* on page 11-341
AFREADYMIx	TRCITIATBOUTR	[1]	See *Integration Instruction ATB Out Register* on page 11-342
ATBYTESMDx[2:0]	TRCITDATBOUTR	[10:8]	See *Integration Data ATB Out Register* on page 11-341
ATBYTESMIx[1:0]	TRCITIATBOUTR	[9:8]	See *Integration Instruction ATB Out Register* on page 11-342
ATDATAMDx[63, 55, 47, 39, 31, 23, 15, 7, 0]	TRCITDDATAR	[8:0]	See *Integration Data ATB Data Register* on page 11-337
ATDATAMIx[31, 23, 15, 7, 0]	TRCITIDATAR	[4:0]	See *Integration Instruction ATB Data Register* on page 11-338
ATIDMDx[6:0]	TRCITATBIDR	[6:0]	See *Integration ATB Identification Register* on page 11-336
ATIDMIx[6:0]	TRCITATBIDR	[6:0]	See *Integration ATB Identification Register* on page 11-336
ATVALIDMDx	TRCITDATBOUTR	[0]	See *Integration Data ATB Out Register* on page 11-341
ATVALIDMIx	TRCITIATBOUTR	[0]	See *Integration Instruction ATB Out Register* on page 11-342
ETMACTIVEx	TRCITMISCOUTR	[5]	See *Integration Miscellaneous Outputs Register* on page 11-334
ETMEXTOUT[3:0]	TRCITMISCOUTR	[11:8]	See *Integration Miscellaneous Outputs Register* on page 11-334

Input signals that the Integration Test Registers can read

Table 11-85 Input signals that the Integration Test Registers can read

Signal	Register	Bits	Register description
AFVALIDMDx	TRCITDATBINR	[1]	See *Integration Data ATB In Register* on page 11-339
ATREADYMDx	TRCITDATBINR	[0]	See *Integration Data ATB In Register* on page 11-339
AFVALIDMIx	TRCITIATBINR	[1]	See *Integration Instruction ATB In Register* on page 11-340
ATREADYMIx	TRCITIATBINR	[0]	See *Integration Instruction ATB In Register* on page 11-340
CPUACTIVE	TRCITMISCINR	[4]	See *Integration Miscellaneous Inputs Register* on page 11-335
DBGACK	TRCITMISCINR	[5]	See *Integration Miscellaneous Inputs Register* on page 11-335
ETMEVENT[3:0]	TRCITMISCINR	[3:0]	See *Integration Miscellaneous Inputs Register* on page 11-335

Using the Integration Test Registers

You must not attempt to write to an Integration Test Register unless you have set bit[0] of TRCITCTRL to `0b1`.

When bit[0] of TRCITCTRL is set to `0b1`:

- Values written to the write-only integration test registers map onto the specified outputs of the macrocell. For example, writing `0x3` TRCITMISCOUTR[11:8] causes **ETMEXTOUT[3:0]** to take the value `0x3`.
- Values read from the read-only integration test registers correspond to the values of the specified inputs of the macrocell. For example, if you read TRCITMISCINR[3:0] you obtain the value of **ETMEXTIN[3:0]**.

When bit[0] of TRCITCTRL is set to `0b0`:

- Reading an Integration Test Register returns an UNPREDICTABLE value.
- The effect of attempting to write to an Integration Test Register, other than the read-only Integration Test Registers, is UNPREDICTABLE.

The *Arm® Cortex®-R8 MPCore Processor Integration Manual* gives a full description of the use of the Integration Test Registers to check integration.

Integration Miscellaneous Outputs Register

The TRCITMISCOUTR sets the state of the output pins for **ETMEXTOUT[3:0]** and **ETMACTIVEx**.

Usage constraints

- Available when bit[0] of TRCITCTRL is set to `0b1`.
- The value of the register sets the signals on the output pins when the register is written.

Configurations

Available in all configurations.

Attributes

Register number: 951

Base offset `0xEDC`

Name: TRCITMISCOUTR

Type: RW

Reset: -

The following figure shows the TRCITMISCOUTR bit assignments.

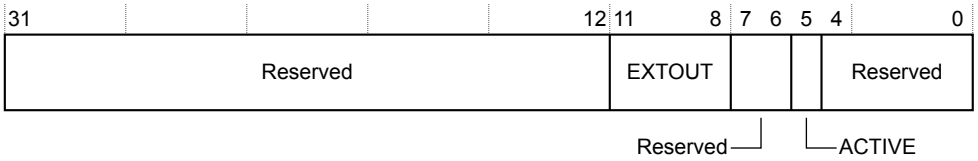


Figure 11-72 TRCITMISCOUTR bit assignments

The following table shows the TRCITMISCOUTR bit assignments.

Table 11-86 TRCITMISCOUTR bit assignments

Bits	Name	Function
[31:12]	-	Reserved. Write as zero.
[11:8]	EXTOUT	Drives the **ETMEXTOUT[3:0]** output pins[bq].
[7:6]	-	Reserved. Write as zero.
[5]	ACTIVE	Drives the **ETMACTIVEx** output pin[bq].
[4:0]	-	Reserved. Write as zero.

Related reference

11.7.1 Cortex®-R8 processor ETM register summary on page 11-254

Output signals that the Integration Test Registers can control on page 11-332

[bq] When a bit is set to `0b0`, the corresponding output pin is LOW. When a bit is set to `0b1`, the corresponding output pin is HIGH. The TRCITMISCOUTR bit values correspond to the physical state of the output pins.

Integration Miscellaneous Inputs Register

The TRCITMISCINR reads the state of the input pins for the **DBGACK**, **CPUACTIVE**, and **ETMEVENT[3:0]** signals.

Usage constraints

- Available when bit[0] of TRCITCTRL is set to `0b1`.
- The values of the register bits depend on the signals on the input pins when the register is read.

Configurations

Available in all configurations.

Attributes

Register number: 952

Base offset `0xEE0`

Name: TRCITMISCINR

Type: RO

Reset: -

The following figure shows the TRCITMISCINR bit assignments.

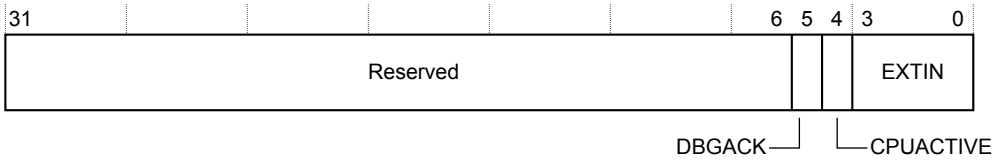


Figure 11-73 TRCITMISCINR bit assignments

The following table shows the TRCITMISCINR bit assignments.

Table 11-87 TRCITMISCINR bit assignments

Bits	Name	Function
[31:6]	-	Reserved. Read UNDEFINED.
[5]	DBGACK	Returns the value of the **DBGACK** input pin[br].
[4]	CPUACTIVE	Returns the value of the **CPUACTIVE** input pin[br].
[3:0]	EXTIN	Returns the value of the **ETMEVENT[3:0]** input pins[br].

Related reference

11.7.1 Cortex®-R8 processor ETM register summary on page 11-254

Input signals that the Integration Test Registers can read on page 11-332

[br] When an input pin is LOW, the corresponding register bit is `0b0`. When an input pin is HIGH, the corresponding register bit is `0b1`. The TRCITMISCINR bit values always correspond to the physical state of the input pins.

Integration ATB Identification Register

The TRCITATBIDR sets the state for the **ATIDMDx[6:0]** and **ATIDMIx[6:0]** output pins.

Usage constraints

- Available when bit[0] of TRCITCTRL is set to `0b1`.
- The value of the register sets the signals on the output pins when the register is written.

Configurations

Available in all configurations.

Attributes

Register number: 953

Base offset `0xEE4`

Name: TRCITATBIDR

Type: RW

Reset: -

The following figure shows the TRCITATBIDR bit assignments.

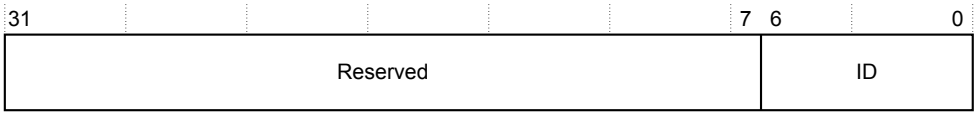


Figure 11-74 TRCITATBIDR bit assignments

The following table shows the TRCITATBIDR bit assignments.

Table 11-88 TRCITATBIDR bit assignments

Bits	Name	Function
[31:7]	-	Reserved. Read UNDEFINED.
[6:0]	ID	Drives the **ATIDMDx[6:0]** and **ATIDMIx[6:0]** output pins[bs].

Related reference

11.7.1 Cortex®-R8 processor ETM register summary on page 11-254

Integration test registers on page 11-264

Output signals that the Integration Test Registers can control on page 11-332

[bs] Bits[6:1] drive both **ATIDMDx[6:1]** and **ATIDMIx[6:1]**. Bit[0] drives **ATIDMIx[0]**. When a bit is set to `0b0`, the corresponding output pin is LOW. When a bit is set to `0b1`, the corresponding output pin is HIGH. **ATIDMDx[0]** is always driven HIGH. The TRCITATBIDR bit values correspond to the physical state of the output pins.

Integration Data ATB Data Register

The TRCITDDATAR sets the state of the **ATDATAMDx[63, 55, 47, 39, 31, 23, 15, 7, 0]** output pins.

Usage constraints

- Available when bit[0] of TRCITCTRL is set to `0b1`.
- The value of the register sets the signals on the output pins when the register is written.

Configurations

Available in all configurations.

Attributes

Register number: 954

Base offset `0xEE8`

Name: TRCIRDDATAR

Type: RW

Reset: -

The following figure shows the TRCITDDATAR bit assignments.

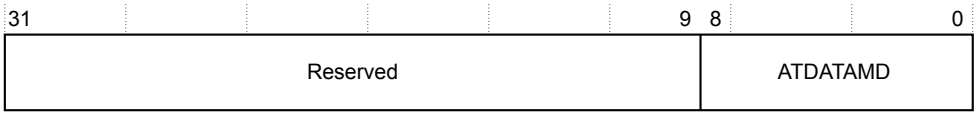


Figure 11-75 TRCITDDATAR bit assignments

The following table shows the TRCITDDATAR bit assignments.

Table 11-89 TRCITDDATAR bit assignments

Bits	Name	Function
[31:9]	-	Reserved. Write as zero.
[8:0]	ATDATAMD	Drives the **ATDATAMDx[63, 55, 47, 39, 31, 23, 15, 7, 0]** output pins[bt].

Related reference

11.7.1 Cortex®-R8 processor ETM register summary on page 11-254

Integration test registers on page 11-264

Output signals that the Integration Test Registers can control on page 11-332

[bt] When a bit is set to `0b0`, the corresponding output pin is LOW. When a bit is set to `0b1`, the corresponding output pin is HIGH. The TRCITDDATAR bit values correspond to the physical state of the output pins.

Integration Instruction ATB Data Register

The TRCITIDATAR sets the state of the **ATDATAMIx[31, 23, 15, 7, 0]** output pins.

Usage constraints

- Available when bit[0] of TRCITCTRL is set to `0b1`.
- The value of the register sets the signals on the output pins when the register is written.

Configurations

Available in all configurations.

Attributes

Register number: 955

Base offset `0xEEC`

Name: TRCITIDATAR

Type: RW

Reset: -

The following figure shows the TRCITIDATAR bit assignments.

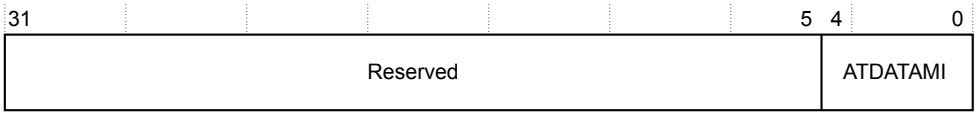


Figure 11-76 TRCITIDATAR bit assignments

The following table shows the TRCITIDATAR bit assignments.

Table 11-90 TRCITIDATAR bit assignments

Bits	Name	Function
[31:5]	-	Reserved. Write as zero.
[4:0]	ATDATAMI	Drives the **ATDATAMIx[31, 23, 15, 7, 0]** output pins[bu].

Related reference

11.7.1 Cortex®-R8 processor ETM register summary on page 11-254

Integration test registers on page 11-264

Output signals that the Integration Test Registers can control on page 11-332

[bu] When a bit is set to `0b0`, the corresponding output pin is LOW. When a bit is set to `0b1`, the corresponding output pin is HIGH. The TRCITIDATAR bit values correspond to the physical state of the output pins.

Integration Data ATB In Register

The TRCITDATBINR reads the state of the **AFVALIDMDx** input pin.

Usage constraints

- Available when bit[0] of TRCITCTRL is set to `0b1`.
- The values of the register bits depend on the signals on the input pins when the register is read.

Configurations

Available in all configurations.

Attributes

Register number: 956

Base offset `0xEF0`

Name: TRCITDATBINR

Type: RO

Reset: -

The following figure shows the TRCITDATBINR bit assignments.

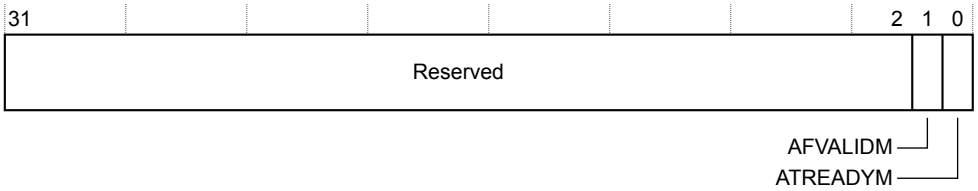


Figure 11-77 TRCITDATBINR bit assignments

The following table shows the TRCITDATBINR bit assignments.

Table 11-91 TRCITDATBINR bit assignments

Bits	Name	Function
[31:2]	-	Reserved. Read UNDEFINED.
[1]	AFVALIDM	Returns the value of the **AFVALIDMDx** input pin[bv].
[0]	ATREADYM	Returns the value of the **ATREADYMDx** input pin[bv].

Related reference

11.7.1 Cortex®-R8 processor ETM register summary on page 11-254

Integration test registers on page 11-264

Input signals that the Integration Test Registers can read on page 11-332

[bv] When an input pin is LOW, the corresponding register bit is `0b0`. When an input pin is HIGH, the corresponding register bit is `0b1`. The TRCITDATBINR bit values always correspond to the physical state of the input pins.

Integration Instruction ATB In Register

The TRCITIATBINR reads the state of the **AFVALIDMIx** and **ATREADYMIx** input pins.

Usage constraints

- Available when bit[0] of TRCITCTRL is set to `0b1`.
- The values of the register bits depend on the signals on the input pins when the register is read.

Configurations

Available in all configurations.

Attributes

Register number: 957

Base offset `0xEF4`

Name: TRCITIATBINR

Type: RO

Reset: -

The following figure shows the TRCITIATBINR bit assignments.

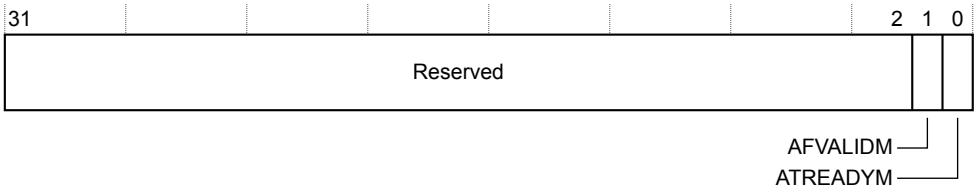


Figure 11-78 TRCITIATBINR bit assignments

The following table shows the TRCITIATBINR bit assignments.

Table 11-92 TRCITIATBINR bit assignments

Bits	Name	Function
[31:2]	-	Reserved. Read UNDEFINED.
[1]	AFVALIDM	Returns the value of the **AFVALIDMIx** input pin.
[0]	ATREADYM	Returns the value of the **ATREADYMIx** input pin[bw].

Related reference

11.7.1 Cortex®-R8 processor ETM register summary on page 11-254

Integration test registers on page 11-264

Input signals that the Integration Test Registers can read on page 11-332

[bw] When an input pin is LOW, the corresponding register bit is `0b0`. When an input pin is HIGH, the corresponding register bit is `0b1`. The TRCITIATBINR bit values always correspond to the physical state of the input pins.

Integration Data ATB Out Register

The TRCITDATBOUTR sets the state of the **ATBYTESMDx[2:0]**, **AFREADYMDx**, and **ATVALIDMDx** output pins.

Usage constraints

- Available when bit[0] of TRCITCTRL is set to `0b1`.
- The value of the register sets the signals on the output pins when the register is written.

Configurations

Available in all configurations.

Attributes

Register number: 958

Base offset `0xEF8`

Name: TRCITDATBOUTR

Type: RW

Reset: -

The following figure shows the TRCITDATBOUTR bit assignments.

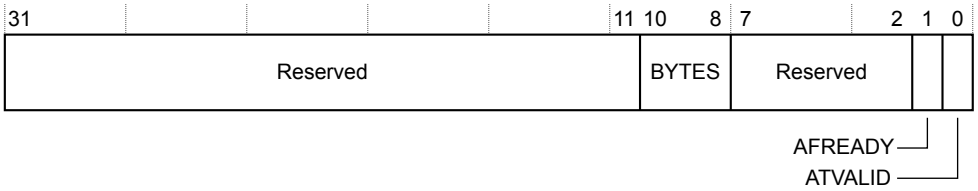


Figure 11-79 TRCITDATBOUTR bit assignments

The following table shows the TRCITDATBOUTR bit assignments.

Table 11-93 TRCITDATBOUTR bit assignments

Bits	Name	Function
[31:11]	-	Reserved. Read UNDEFINED.
[10:8]	BYTES	Drives the **ATBYTESMDx[2:0]** output pins.
[7:2]	-	Reserved. Read UNDEFINED.
[1]	AFREADY	Drives the **AFREADYMDx** output pin[bx].
[0]	ATVALID	Drives the **ATVALIDMDx** output pin[bx].

Related reference

11.7.1 Cortex®-R8 processor ETM register summary on page 11-254

Integration test registers on page 11-264

Output signals that the Integration Test Registers can control on page 11-332

[bx] When a bit is set to `0b0`, the corresponding output pin is LOW. When a bit is set to `0b1`, the corresponding output pin is HIGH. The TRCITDATBOUTR bit values always correspond to the physical state of the output pins.

Integration Instruction ATB Out Register

The TRCITIATBOUTR sets the state of the **ATBYTESMIx[1:0]**, **AFREADYMIx**, and **ATVALIDMIx** output pins.

Usage constraints

- Available when bit[0] of TRCITCTRL is set to `0b1`.
- The value of the register sets the signals on the output pins when the register is written.

Configurations

Available in all configurations.

Attributes

Register number: 959

Base offset `0xEFC`

Name: TRCITIATBOUTR

Type: RW

Reset: -

The following figure shows the TRCITIATBOUTR bit assignments.

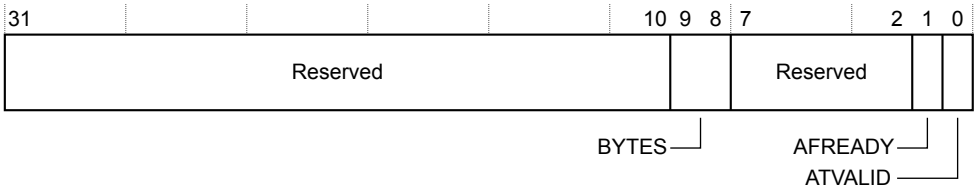


Figure 11-80 TRCITIATBOUTR bit assignments

The following table shows the TRCITIATBOUTR bit assignments.

Table 11-94 TRCITIATBOUTR bit assignments

Bits	Name	Function
[31:10]	-	Reserved. Read UNDEFINED.
[9:8]	BYTES	Drives the **ATBYTESMIx[1:0]** output pins.
[7:2]	-	Reserved. Read UNDEFINED.
[1]	AFREADY	Drives the **AFREADYMIx** output pin[by].
[0]	ATVALID	Drives the **ATVALIDMIx** output pin[by].

Related reference

11.7.1 Cortex®-R8 processor ETM register summary on page 11-254

Integration test registers on page 11-264

Output signals that the Integration Test Registers can control on page 11-332

Related reference

11.8.49 Integration Mode Control Register on page 11-319

[by] When a bit is set to `0b0`, the corresponding output pin is LOW. When a bit is set to `0b1`, the corresponding output pin is HIGH. The TRCITIATBOUTR bit values always correspond to the physical state of the output pins.

Chapter 12
Level 2 Interface

This chapter describes the L2 memory interface.

It contains the following sections:

12.1 About the L2 interface

The Cortex-R8 processor L2 interface consists of one or two 64-bit wide AXI3 bus masters. AXI3 M0 is always present. AXI3 M1 is optional and supports address filtering.

The following table shows the AXI3 master 0 and master 1 interface attributes.

Table 12-1 AXI3 master 0 and master 1 interface attributes

Attribute	Format
Write issuing capability	29 + 12 * CN[bz] for an implementation with CN + 1 cores, including: • 15 ACP writes. • For each core: — Eight Non-Cacheable writes. — Four evictions. • Two coherency operation evictions.
Read issuing capability	31 + 16 * CN[bz] for an implementation with CN + 1 cores, including: • 15 ACP reads. • For each core: — Four data side linefill reads. — Eight instruction side linefill reads. — Four Non-Cacheable reads.
Combined issuing capability	60 + 28 * CN[bz].
Write interleave capability	1.

Note

The numbers given in the table are the theoretical maximums for the Cortex-R8 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 *Arm AMBA® AXI and ACE Protocol Specification AXI3, AXI4, and AXI4-Lite, ACE and ACE-Lite*.

This section contains the following subsections:

• *12.1.1 Supported AXI3 transfers* on page 12-344.
• *12.1.2 AXI3 USER bits* on page 12-345.

12.1.1 Supported AXI3 transfers

The Cortex-R8 processor master ports can generate all possible AXI3 transactions from ACP traffic, including transaction types that differ from the original ACP transaction. ACP transactions can trigger Wrapping burst write transactions on the main AXI interface.

Transactions from the individual cores 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.

[bz] CN represents the number of configured cores minus one.

— 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, four 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.

12.1.2 AXI3 USER bits

AXI3 USER bits encodings.

Read address channel of AXI master 0, ARUSERM0[9:0]

Bit encodings for **ARUSERM0[9:0]**

Table 12-2 ARUSERM0[9:0] encodings

Bits	Name	Description	
[9:7]	Transaction type	`0b000`	Core 0 transaction
		`0b010`	Core 1 transaction
		`0b100`	Core 2 transaction
		`0b110`	Core 3 transaction
		`0b001`	ACP transaction
[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[ca]
		`0b0111`	Write-Back no Write-Allocate
		`0b1111`	Write-Back Write-Allocate
[0]	Shareable bit	`0b0`	Non-Shareable
		`0b1`	Shareable

Read address channel of AXI master 1, ARUSERM1[9:0]

Bit encodings for **ARUSERM1[9:0]**.

[ca] If Write-Through is used in the MPU, it behaves as normal memory, Non-Cacheable, and its value is `0b0110`.

Table 12-3 ARUSERM1[9:0] encodings

Bits	Name	Description	
[9:7]	Transaction type	`0b000`	Core 0 transaction
		`0b010`	Core 1 transaction
		`0b100`	Core 2 transaction
		`0b110`	Core 3 transaction
		`0b001`	ACP transaction
[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[cb]
		`0b0111`	Write-Back no Write-Allocate
		`0b1111`	Write-Back Write-Allocate
[0]	Shareable bit	`0b0`	Non-Shareable
		`0b1`	Shareable

Read address bus of AXI low-latency peripheral, ARUSERMP[9:0]

Bit encodings for **ARUSERMP[9:0]**.

Table 12-4 ARUSERMP[9:0] encodings

Bits	Name	Description	
[9:7]	Transaction type	`0b000`	Core 0 transaction
		`0b010`	Core 1 transaction
		`0b100`	Core 2 transaction
		`0b110`	Core 3 transaction
		`0b001`	ACP transaction
[6:5]	Reserved	`0b00`	
[4:1]	Inner attributes	`0b0000`	Strongly Ordered
		`0b0001`	Device
		`0b0011`	Normal Memory Non-Cacheable
		`0b0110`	Reserved[cc]
		`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[11:0]

Bit encodings for **AWUSERM0[11:0]**.

[cb] If Write-Through is used in the MPU, it behaves as normal memory, Non-Cacheable, and its value is `0b0110`.
[cc] If Write-Through is used in the MPU, it behaves as normal memory, Non-Cacheable, and its value is `0b0110`.

Table 12-5 AWUSERM0[11:0] encodings

Bits	Name	Description	
[11:9]	Transaction type	`0b000`	Core 0 transaction.
		`0b010`	Core 1 transaction.
		`0b100`	Core 2 transaction.
		`0b110`	Core 3 transaction.
		`0b001`	ACP transaction.
[8]	Early BRESP Enable bit	Indicates that the L2 slave can send an early BRESP answer to the write request. See *12.2.1 Early BRESP* on page 12-350.	
[7:5]	Reserved	RAZ.	
[4:1]	Inner attributes	`0b0000`	Strongly Ordered.
		`0b0001`	Device.
		`0b0011`	Normal Memory Non-Cacheable.
		`0b0110`	Reserved.[cd]
		`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 1, AWUSERM1[11:0]

Bit encodings for **AWUSERM1[11:0]**.

Table 12-6 AWUSERM1[11:0] encodings

Bits	Name	Description	
[11:9]	Transaction type	`0b000`	Core 0 transaction.
		`0b010`	Core 1 transaction.
		`0b100`	Core 2 transaction.
		`0b110`	Core 3 transaction.
		`0b001`	ACP transaction.
[8]	Early BRESP Enable bit	Indicates that the L2 slave can send an early BRESP answer to the write request. See *12.2.1 Early BRESP* on page 12-350.	
[7:5]	Reserved	RAZ.	
[4:1]	Inner attributes	`0b0000`	Strongly Ordered.
		`0b0001`	Device.
		`0b0011`	Normal Memory Non-Cacheable.
		`0b0110`	Reserved.[ce]
		`0b0111`	Write-Back no Write-Allocate.
		`0b1111`	Write-Back Write-Allocate.
[0]	Shareable bit	`0b0`	Non-Shareable.
		`0b1`	Shareable.

[cd] If Write-Through is used in the MPU, it behaves as normal memory, Non-Cacheable, and its value is `0b0110`.
[ce] If Write-Through is used in the MPU, it behaves as normal memory, Non-Cacheable, and its value is `0b0110`.

Write address bus of AXI low-latency peripheral, AWUSERMP[11:0]

Bit encodings for **AWUSERMP[11:0]**.

Table 12-7 AWUSERMP[11:0] encodings

Bits	Name	Description	
[11:9]	Transaction type	`0b000`	Core 0 transaction
		`0b010`	Core 1 transaction
		`0b100`	Core 2 transaction
		`0b110`	Core 3 transaction
		`0b001`	ACP transaction
[8:5]	Reserved	RAZ	
[4:1]	Inner attributes	`0b0000`	Strongly Ordered
		`0b0001`	Device
		`0b0011`	Normal Memory Non-Cacheable
		`0b0110`	Reserved[cf]
		`0b0111`	Write-Back no Write-Allocate
		`0b1111`	Write-Back Write-Allocate
[0]	Shareable bit	`0b0`	Non-Shareable
		`0b1`	Shareable

Write data channel of AXI master 0, WUSERM0[2:0]

Bit encodings for **WUSERM0[2:0]**.

Table 12-8 WUSERM0[2:0] encodings

Bits	Name	Description	
[2:0]	Transaction type	`0b000`	Core 0 transaction
		`0b010`	Core 1 transaction
		`0b100`	Core 2 transaction
		`0b110`	Core 3 transaction
		`0b001`	ACP transaction

Write data channel of AXI master 1, WUSERM1[2:0]

Bit encodings for **WUSERM1[2:0]**.

Table 12-9 WUSERM1[2:0] encodings

Bits	Name	Description	
[2:0]	Transaction type	`0b000`	Core 0 transaction
		`0b010`	Core 1 transaction
		`0b100`	Core 2 transaction
		`0b110`	Core 3 transaction
		`0b001`	ACP transaction

[cf] If Write-Through is used in the MPU, it behaves as normal memory, Non-Cacheable, and its value is `0b0110`.

Write data bus of AXI low-latency peripheral, WUSERMP[2:0]

Bit encodings for **WUSERMP[2:0]**.

Table 12-10 WUSERMP[2:0] encodings

Bits	Name	Description	
[1:0]	Transaction type	`0b000`	Core 0 transaction
		`0b010`	Core 1 transaction
		`0b100`	Core 2 transaction
		`0b110`	Core 3 transaction
		`0b001`	ACP transaction

12.2 Optimized accesses to the L2 memory interface

Optimized accesses to the L2 memory interface can generate non-AXI3 compliant requests on the AXI master ports. These non-AXI 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.

12.2.1 Early BRESP

According to the AXI3 specification, **BRESP** answers on response channels must be returned to the master only after the master sends the last data. Cortex-R8 processor cores can also deal with **BRESP** answers returned when the address is accepted by the slave, regardless of whether data is sent or not.

This enables the core to provide a higher bandwidth for writes if the slave can support the Early **BRESP** feature. Cortex-R8 processor cores 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 core, 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-R8 processor core does not require any programming to enable this feature.

Note

To support this optimization, you must program the L2C-310 Cache Controller. See the *Arm® CoreLink™ Level 2 Cache Controller L2C-310 Technical Reference Manual*.

12.2.2 SCU speculative coherent requests

This optimization is available for Cortex-R8 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 speculative request already initiated the data request. When filtering is enabled, only port 0 can receive speculative linefills.

To support this optimization in the Cortex-R8 processor:

1. Program the L2C-310 Cache Controller. See the *Arm® CoreLink™ Level 2 Cache Controller L2C-310 Technical Reference Manual*.
2. Set bit[3] of the SCU Control Register.

Note

You cannot use this feature when bus ECC is implemented and the L2C-310 Cache Controller is connected.

Related reference

9.3.1 SCU Control Register on page 9-169

12.3 Accessing RAMs using the AXI3 interface

The Cortex-R8 processor has a single AXI TCM 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** each as two separate chip select input signals to enable access to the TCMs as shown in the following table.

Table 12-11 TCM accesses

AxUSERST[2:0] value	TCM
0b000	Instruction TCM of core 0
0b001	Data TCM of core 0
0b010	Instruction TCM of core 1
0b011	Data TCM of core 1
0b100	Instruction TCM of core 2
0b101	Data TCM of core 2
0b110	Instruction TCM of core 3
0b111	Data TCM of core 3

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

The following table shows the MSB bit for the different TCM RAM sizes.

Table 12-12 MSB bit for the different TCM RAM sizes

TCM size	ARADDRST[MSB]
4KB	[11]
8KB	[12]
16KB	[13]
32KB	[14]
64KB	[15]
128KB	[16]
256KB	[17]
512KB	[18]
1024KB	[19]

ARADDRST[19:3] 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 1024KB, 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 an SLVERR error response if:

- It is outside the physical size of the targeted TCM RAM, that is, bits of **ARADDRST[19:MSB+1]** are nonzero.
- There is no TCM present. The mapping of bus addresses to **ARUSERST** and **ARADDRST** is determined when the processor is integrated. You must understand this mapping to use the AXI TCM slave interface in your system.

12.4 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.

The following table shows Cortex-R8 processor modes and corresponding **AxPROT** values.

Table 12-13 Cortex-R8 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

12.5 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-R8 processor by using the **EVENTI** input.

When this input is asserted, it sends an event message to all the cores in the design. This is similar to executing a `SEV` instruction on one core of the Cortex-R8 processor. This enables the external agent to signal to the cores that it has released a semaphore and that the cores can leave the power-saving mode. The **EVENTI** input must remain HIGH at least one **CLK** clock cycle to be visible by the cores.

The external agent can see that at least one of the cores has executed an `SEV` instruction by checking the **EVENTO** output. This output is set HIGH for one **CLK** clock cycle when any of the cores executes an `SEV` instruction.

Related concepts

2.4 Power management on page 2-36

12.6 Accelerator Coherency Port interface

The optional *Accelerator Coherency Port* (ACP) provides memory coherency between each core in the Cortex-R8 processor design and an external master.

The ACP is 64 bits wide, and conforms to the AMBA 3 AXI standard as described in the *Arm® AMBA® AXI and ACE Protocol Specification AXI3, AXI4, and AXI4-Lite, ACE and ACE-Lite*.

12.6.1 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 **ARUSERSC[0]** = `0b1` and **ARCACHESC[1]** = `0b1`, and **ARVALIDSC** is asserted.

In this case, the SCU enforces coherency.

When the data is present in one of the Cortex-R8 processor cores, the data is read directly from the relevant core, and returned to the ACP port.

When the data is not present in any of the cores, the read request is issued on one of the AXI3 master ports, with all its AXI parameters, except for the locked attribute.

ACP noncoherent read requests

An ACP read request is noncoherent when **ARUSERSC[0]** = `0b0` or **ARCACHESC[1]** = `0b0`, and **ARVALIDSC** 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 **AWUSERSC[0]** = `0b1` and **AWCACHESC[1]** = `0b1`, and **AWVALIDSC** is asserted.

In this case, the SCU enforces coherency.

When the data is present in one of the Cortex-R8 processor cores, the data is first cleaned and invalidated from the relevant core.

When the data is not present in any of the cores, 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 noncoherent write requests

An ACP write request is noncoherent when **AWUSERSC[0]** = `0b0` or **AWCACHESC[1]** = `0b0`, and **AWVALIDSC** 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.

12.6.2 ACP limitations

The ACP is optimized for cache-line length transfers and supports a wide range of AMBA3 AXI3 requests, but it has some performance and functional limitations that must be considered.

ACP performance limitations

ACP accesses are optimized for transfers that match Cortex-R8 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.

Related reference

9.7.6 ACP bridge on page 9-212

ACP functional limitations

ACP unsupported transfers.

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 **AxUSERSC[0]** = `0b1` and **AxCACHESC[1]** = `0b1`.

However, the LDREX/STREX mechanism is fully supported through the ACP for noncoherent memory regions, marked with **AxUSERSC[0]** = `0b0` or **AxCACHESC[1]** = `0b0`.

64-bit accesses to the AXI low-latency peripheral port always abort. 32-bit wide normal memory Non-Cacheable accesses from the ACP to the AXI low-latency peripheral port do not abort.

Related concepts

2.5.3 AXI low-latency peripheral port on page 2-43

Related reference

9.7.6 ACP bridge on page 9-212

Appendix A
Signal Descriptions

This appendix describes the Cortex-R8 processor signals. Signal name, direction, and source/destination details are provided for each signal.

It contains the following sections:

A.1 About the signal descriptions

The tables in this appendix list the Cortex-R8 processor signals, with their direction, either input or output, and a high-level description.

Unless stated otherwise, the signals in this section have the following formats:

- <**signal name**>**[**<**value**>**:0]** where **[**<**value**>**:0**] is the signal bit range, for example **DBGROMADDR[31:12]**.
- <**signal name**>**[CN:0]**, where **CN** represents the number of configured cores minus one, for example **DBGACK[CN:0]**, if the number of configured cores is 2, **CN** = **1**.
- <**signal name**>**x**, where **x** represents the core ID number, **0**, **1**, **2** or **3**, for example **FPFILTERSTARTx**.
- <**signal name**>**x[**<**value**>**:0]** where **x** represents the core ID number, **0**, **1**, **2** or **3** and **[**<**value**>**:0]** is the signal bit range, for example **FPUFLAGSx[1:0]**.

There is no link between the number of cores in a design and the number of AXI master ports in a design. A single core design can have one or two AXI master ports.

A.2 Clock and control signals

Details of the Cortex-R8 processor clock and control signals.

Table A-1 Clock and clock control signals

Name	Type	Source/destination	Description
CLK	Input	Clock controller	Global clock.
CPUCLKOFF[CN:0][cg]	Input	Reset controller	Individual core clock control, active-LOW: `0b0` Clock is enabled. `0b1` Clock is stopped. This includes the FPU if it is present.
DBGCLKOFF[CN:0][cg]	Input		Individual core debug clock control, active-LOW: `0b0` Core debug clock is enabled. `0b1` Core debug clock is stopped.
DUALPERIPHCLK[ch]	Input	Clock controller	Clock for dual SCU.
DUALPERIPHCLKEN[ch]	Input	Reset controller	Clock enable for dual SCU and peripheral interface signals.
DUALPERIPHCLKOFF[cgch]	Input		Individual core clock control for timer, watchdog, and interrupt controller of dual SCU.
PERIPHCLK	Input	Clock controller	Clock for the timer, watchdog, and interrupt controller.
PERIPHCLKEN	Input	Reset controller	Clock enable for the timer, watchdog, and interrupt controller.
PERIPHCLKOFF[cgch]	Input		Individual core clock control for timer, watchdog, and interrupt controller of SCU.
SCUCLKOFF[cgch]	Input		Clock control delay for dual SCU.
CTCLKOFF[cg]	Input		Used to control the CoreSight debug logic clock, that is, CTI0, CTI1, CTI2, CTI3, CTM, and ROM table.
ETM0CLKOFF[cg]	Input		Controls the ETM0 clock, if ETM0 is present.
ETM1CLKOFF[cg]	Input		Controls the ETM1 clock, if ETM1 is present.
ETM2CLKOFF[cg]	Input		Controls the ETM2 clock, if ETM2 is present.
ETM3CLKOFF[cg]	Input		Controls the ETM3 clock, if ETM3 is present.

[cg] Deasserts the reset synchronously when leaving reset, but not used for clock enable.
[ch] Only present if lock-step or split/lock is implemented. The figure shows how these signals are used in a lock-step or split/lock implementation.

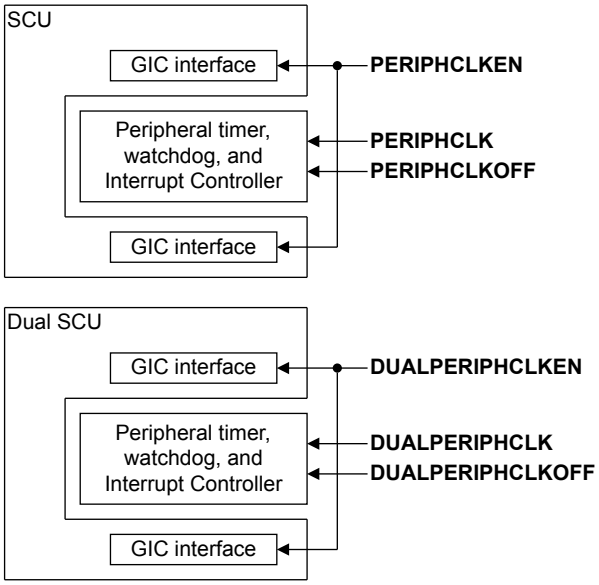


Figure A-1 Clocking in lock-step or split/lock implementation

Related concepts

2.3 Clocking, resets, and initialization on page 2-29

A.3 Reset signals

Details of the Cortex-R8 processor reset signals.

Table A-2 Reset signals

Name	Type	Source/destination	Description
nCPURESET[CN:0]	Input	Reset controller	Individual core resets.
nCPUHALT[CN:0]	Input		Individual core input. It can be asserted while the core is in reset to stop the core from fetching and executing instructions after coming out of reset. While the core is halted in this way, the TCMs can be preloaded with the appropriate data. When it is deasserted, the core starts fetching instructions from the reset vector address in the normal way.
nDBGRESET[CN:0]	Input		Core debug logic resets.
nPERIPHRESET	Input		Timer and interrupt controller reset.
nSCURESET	Input		SCU global reset.
nCTRESET	Input		Reset for CoreSight debug logic, that is, CTI0, CTI1, CTI2, CTI3, CTM, and ROM table.
nETM0RESET	Input		Reset for ETM0, if present.
nETM1RESET	Input		Reset for ETM1, if present.
nETM2RESET	Input		Reset for ETM2, if present.
nETM3RESET	Input		Reset for ETM3, if present.
nWDRESET[CN:0]	Input		Core watchdog resets.
WDRESETREQ[CN:0]	Output		Core watchdog reset requests.

Related concepts

2.3.2 Resets on page 2-29

A.4 Interrupt controller signals

Details of the Cortex-R8 processor interrupt controller signals.

Table A-3 Interrupt controller signals

Name	Type	Source/ destination	Description
IRQS[m:0]	Input	Interrupt sources	Interrupt distributor interrupt lines. **m** can be 31, 63,…, up to 479 in increments of 32. If there are no interrupt lines, this input is removed.
nFIQ[CN:0]	Input		Individual core legacy FIQ request input lines. Active-LOW interrupt request: `0b0` Active interrupt. `0b1` Do not activate interrupt. The core treats the **nFIQ** input as level sensitive. The **nFIQ** input must be asserted until the core acknowledges the interrupt.
nFIQOUT[CN:0]	Output	Power controller	Active-LOW **FIQ** outputs from the internal GIC to the appropriate core. These indicate when interrupts are being forwarded to the core.
nIRQ[CN:0]	Input	Interrupt sources	Individual core legacy IRQ request input lines. Active-LOW interrupt request: `0b0` Active interrupt. `0b1` Do not activate interrupt. The core treats the **nIRQ** input as level sensitive. The **nIRQ** input must be asserted until the core acknowledges the interrupt.
nIRQOUT[CN:0]	Output	Power controller	Active-LOW **IRQ** outputs from the internal GIC to the appropriate core. These indicate when interrupts are being forwarded to the core.

A.5 Configuration signals

Details of the Cortex-R8 processor configuration signals.

Table A-4 Configuration signals

Name	Type	Source/ destination	Description
AXIPARITYLEVEL[ci]	Input	System configuration	Selects between odd and even parity for buses: `0b0` Even parity. `0b1` Odd parity.
CFGEND[CN:0]	Input		Individual core endianness configuration. Forces the EE bit in the CP15 c1 Control Register (SCTLR) to 1 at reset so that the core boots with big-endian data handling: `0b0` EE bit is LOW. `0b1` EE bit is HIGH. This input is only sampled during reset of the core. See *4.3.9 System Control Register* on page 4-77.
CFGNMFI[CN:0]	Input		Configuration of FIs to be nonmaskable: `0b0` Clear the NMFI bit in the CP15 c1 Control Register. `0b1` Set the NMFI bit in the CP15 c1 Control Register. This input is only sampled during reset of the core. See *4.3.9 System Control Register* on page 4-77.
CLUSTERID[3:0]	Input		Value read in Cluster ID field, bits[11:8], of the *Multiprocessor Affinity Register* (MPIDR). See *4.3.3 Multiprocessor Affinity Register* on page 4-72.
INITRAMx	Input		Input present if TCM present for the corresponding core. It enables the core to boot from the Instruction TCM. This input, when tied HIGH, enables the instruction TCM on leaving reset. See *4.3.14 ITCM Region Register* on page 4-89.
MFILTEREN	Input		For configurations with two master ports. 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 *9.3.1 SCU Control Register* on page 9-169. `0b0` Address filtering off. `0b1` Address filtering on. See *9.3.1 SCU Control Register* on page 9-169 and *2.5.2 AXI master port 1* on page 2-42.
MFILTEREND[11:0]	Input		For configurations with two master ports. Specifies the end address for address filtering at reset on AXI master port 1. See *9.3.1 SCU Control Register* on page 9-169 and *2.5.2 AXI master port 1* on page 2-42.

[ci] Only present if bus ECC is selected. This is a build option.

Table A-4 Configuration signals (continued)

Name	Type	Source/ destination	Description
MFILTERSTART[11:0]	Input	System configuration	For configurations with two master ports. Specifies the start address for address filtering at reset on AXI master port 1. See *9.3.1 SCU Control Register* on page 9-169 and *2.5.2 AXI master port 1* on page 2-42.
PERIPHBASE[31:13]	Input		Specifies the base address for timers, watchdogs, interrupt controller, and SCU registers. Only accessible with memory-mapped accesses. This value can be retrieved by a core using the Configuration Base Address Register. See *4.3.20 Configuration Base Address Register* on page 4-97. ——— **Note** ——— This address must be in the range defined by **PFILTERSTART[11:0]** and **PFILTEREND[11:0]**. ———
PFILTEREND[11:0]	Input		For configurations with the AXI low-latency peripheral port. Specifies the end address for address filtering at reset on the AXI low-latency peripheral port. See *2.5.3 AXI low-latency peripheral port* on page 2-43.
PFILTERSTART[11:0]	Input		For configurations with the AXI low-latency peripheral port. Specifies the start address for address filtering at reset on the AXI low-latency peripheral port. See *2.5.3 AXI low-latency peripheral port* on page 2-43.
FPFILTERENDx[11:0]	Input		For cores configured with an AXI fast peripheral port. Specifies the end address for address filtering at reset on the AXI fast peripheral port. See *2.5.4 AXI Fast Peripheral Port* on page 2-44.
FPFILTERSTARTx[11:0]	Input		For cores configured with an AXI fast peripheral port. Specifies the start address for address filtering at reset on the AXI fast peripheral port. See *2.5.4 AXI Fast Peripheral Port* on page 2-44.
SMPnAMP[CN:0]	Output	System integrity controller	Indicates AMP or SMP mode for each core: `0b0` Asymmetric. `0b1` Symmetric. This output reflects the value of ACTLR.SMP. See *4.3.10 Auxiliary Control Register* on page 4-80.

Table A-4 Configuration signals (continued)

Name	Type	Source/ destination	Description
TEINIT[CN:0]	Input	System configuration	Individual core out-of-reset default exception handling state: `0b0` Arm. `0b1` Thumb. This input is only sampled during core reset. It sets the initial value of SCTLR.TE. See *4.3.9 System Control Register* on page 4-77.
VINITHI[CN:0]	Input		Individual core control of the location of the exception vectors at reset: `0b0` Exception vectors start at address `0x00000000`. `0b1` Exception vectors start at address `0xFFFF0000`.[cj] This input is only sampled during core reset. It sets the initial value of SCTLR.V. See *4.3.9 System Control Register* on page 4-77 and *4.3.14 ITCM Region Register* on page 4-89.

[cj] `HIVECS == 1` is deprecated in PMSAv7.

A.6 Standby signals

Cortex-R8 processor standby signals.

A.6.1 Standby and Wait for event signals

Details of the Cortex-R8 processor Standby and Wait for Event signals.

Table A-5 Standby and Wait for event signals

Name	Type	Source/destination	Description
STANDBYWFI[CN:0]	Output	Wakeup controller	Indicates if the corresponding core is in Standby WFI
STANDBYWFE[CN:0]	Output		Indicates if the corresponding core is in Standby WFE

A.6.2 Event signals

Details of the Cortex-R8 processor 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

Related concepts

Standby modes on page 2-36

Related reference

9.3.1 SCU Control Register on page 9-169

A.7 Power management signals

Details of the Cortex-R8 processor power management signals.

Table A-7 Power management signals

Name	Type	Source/ destination	Description
PWRCTLO0[1:0]	Output	Power controller	Reset value for core 0 status field, bits[1:0] of SCU CPU Power Status Register.
PWRCTLO1[1:0][ck]	Output		Reset value for core 1 status field, bits[9:8] of SCU CPU Power Status Register.
PWRCTLO2[1:0][ck]	Output		Reset value for core 2 status field, bits[17:16] of SCU CPU Power Status Register.
PWRCTLO3[1:0][ck]	Output		Reset value for core 3 status field, bits[25:24] of SCU CPU Power Status Register.
SCUIDLE	Output	L2C-310 or power controller	In the case of the L2C-310, the **SCUIDLE** output can be connected to the **STOPCLK** input of the L2C-310. Indicates the SCU is idle. The SCU is idle when all cores 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 low-latency peripheral port. ——— **Note** ——— When using the ACP, even if there is no activity on the bus, the **ACLKENSC** input must remain HIGH, or must toggle at least once through HIGH, after the activity has stopped. If not, the **SCUIDLE** output cannot go HIGH. ———

Related concepts

2.4.2 Power domains on page 2-41

Related reference

9.3.3 SCU CPU Power Status Register on page 9-173

[ck] Only present if core 1-3 is present.

A.8 AXI3 interfaces

Details of the AXI3 signals, including the AXI master interface signals, AXI fast peripheral port signals, and AXI low-latency peripheral port signals. The AXI3 interfaces are defined in the AMB3 specification.

This section contains the following subsections:

A.8.1 AXI master interface signals

Cortex-R8 processor AXI master interface signals.

The **x** at the end of the signal name represents either **0** for AXI master port 0 or **1** for the optional AXI master port 1.

AXI master interface clock enable signals

Details of the AXI master interface clock enable signals.

Table A-8 AXI master interface clock enable signals

Name	Type	Source/ destination	Description
INCLKENMx	Input	**CLK**	Clock bus enable for the AXI bus that enables the AXI interface to operate at either: • Integer ratios of the system clock. • Half integer ratios of the system clock. Inputs are sampled on rising edges of **CLK** only when **INCLKENMx** is HIGH.
OUTCLKENMx	Input		Clock bus enable for the AXI bus that enables the AXI interface to operate at either: • Integer ratios of the system clock. • Half integer ratios of the system clock. Outputs are updated on rising edges of **CLK** only when **OUTCLKENMx** is HIGH.

AXI master interface read address signals

Details of the AXI master interface read address signals.

Table A-9 AXI master interface read address signals

Name	Type	Source/destination	Description
ARADDRMx[31:0]	Output	AXI3 device	Read address.
ARBURSTMx[1:0]	Output		Read address burst type: `0b01` INCR incrementing burst. `0b10` WRAP wrapping burst. All other values are reserved.
ARCACHEMx[3:0]	Output		Read address cache type giving additional information about cacheable characteristics.
ARIDMx[n][cl]	Output		Read address ID.

Table A-9 AXI master interface read address signals (continued)

Name	Type	Source/destination	Description
ARLENMx[3:0]	Output	AXI3 device	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 *12.1.1 Supported AXI3 transfers* on page 12-344. Burst transactions from the ACP can be 1-16 transfers long. `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		Read address lock type: `0b00` Normal access. `0b01` Exclusive access. `0b10` Locked access.
ARPROTMx[2:0]	Output		Read address protection type
ARREADYMx	Input		Read address ready
ARSIZEMx[1:0]	Output		Read address burst size: `0b000` 8-bit transfer. `0b001` 16-bit transfer. `0b010` 32-bit transfer. `0b011` 64-bit transfer.
ARUSERMx[9:0]	Output		Read address transfer attributes. See *12.1.2 AXI3 USER bits* on page 12-345.
ARVALIDMx	Output		Read address valid.

AXI master interface read data signals

Details of the AXI master interface read data signals.

[cl] 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 [5:0].

Table A-10 AXI master interface read data signals

Name	Type	Source/destination	Description
RDATAMx[63:0]	Input	AXI3 device	Read data.
RDATAERRCODEMx[7:0]	Input		ECC bits on data bus, when `BUS_ECC` build parameter is set.
RIDMx[n]	Input		Read data ID.
RLASTMx	Input		Read data last indication.
RREADYMx	Output		Read data ready.
RRESPMx[1:0]	Input		Read data response.
RVALIDMx	Input		Read data valid.
SRENDMx[3:0]	Input		Speculative read information from optional L2 Cache Controller. See the *Arm® CoreLink™ Level 2 Cache Controller L2C-310 Technical Reference Manual* for more information.
SRIDMx[n][cm]	Input		ID for speculative reads returned by L2 Cache Controller.

AXI master interface write address signals

Details of the AXI master interface write address signals.

Table A-11 AXI master interface write address signals

Name	Type	Source/destination	Description
AWADDRMx[31:0]	Output	AXI3 device	Write address.
AWBURSTMx[1:0]	Output		Write address burst type: `0b01` INCR incrementing burst. `0b10` WRAP wrapping burst. All other values are reserved.
AWCACHEMx[3:0]	Output		Write address cache type giving additional information about cacheable characteristics.
AWIDMx[n][cn]	Output		Write address channel ID.

[cm] 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 [5:0].

[cn] 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 [5:0].

Table A-11 AXI master interface write address signals (continued)

Name	Type	Source/destination	Description
AWLENMx[3:0]	Output	AXI3 device	The number of data transfers that can occur within each burst. For a description of other processor-generated traffic, see *12.1.1 Supported AXI3 transfers* on page 12-344. Burst transactions from the ACP can be 1-16 transfers long. `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		Write address lock type: `0b0` Normal access. `0b1` Exclusive access. `0b10` Locked access.
AWPROTMx[2:0]	Output		Write address protection type.
AWREADYMx	Input		Write address ready.
AWSIZEMx[1:0]	Output		Write address burst size: `0b00` 8-bit transfer. `0b01` 16-bit transfer. `0b10` 32-bit transfer. `0b11` 64-bit transfer.
AWUSERMx[11:0]	Output		Write address transfer attributes. See *12.1.2 AXI3 USER bits* on page 12-345.
AWVALIDMx	Output		Write address valid.

AXI master interface write data signals

Details of the AXI master interface write data signals.

Table A-12 AXI master interface write data signals

Name	Type	Source/destination	Description
WDATAMx[63:0]	Output	AXI3 device	Write data.
WIDMx[n][co]	Output		Write data ID.
WLASTMx	Output		Write last indication.
WREADYMx	Input		Write ready.
WSTRBMx[7:0]	Output		Write byte-lane strobe.
WVALIDMx	Output		Write valid.
WUSERMx[2:0]	Output		Write data transfer attributes. See *12.1.2 AXI3 USER bits* on page 12-345.

AXI master interface write response signals

Details of the AXI master interface write response signals.

Table A-13 AXI master interface write response signals

Name	Type	Source/destination	Description
BIDMx[n][cp]	Input	L2C-310 or other system AXI3 devices	Write response ID
BREADYMx	Output		Write response ready
BRESPMx[1:0]	Input		Write response
BVALIDMx	Input		Write response valid

A.8.2 AXI fast peripheral port signals

The Cortex-R8 processor AXI fast peripheral port signals.

AXI fast peripheral clock enable signals

Details of the AXI fast peripheral clock enable signals.

Table A-14 AXI fast peripheral clock enable signals

Name	Type	Source/destination	Description
INCLKENMFPx	Input	**CLK**	Clock enable
OUTCLKENMFPx	Input		Clock enable

AXI fast peripheral port read address signals

Details of the AXI fast peripheral port read address signals.

[co] 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 [5:0].

[cp] 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 [5:0].

Table A-15 AXI fast peripheral port read address signals

Name	Type	Source/destination	Description
ARADDRMFPx[31:0]	Output	AXI3 fast peripheral port IO	Read address
ARBURSTMFPx[1:0]	Output		Read address burst type
ARCACHEMFPx[3:0]	Output		Read address cache type
ARLENMFPx[3:0]	Output		Read address burst length
ARLOCKMFPx[1:0]	Output		Read address lock type
ARPROTMFPx[2:0]	Output		Read address protection type
ARREADYMFPx	Input		Read address ready
ARSIZEMFPx[1:0]	Output		Read address burst size
ARVALIDMFPx	Output		Read address valid

AXI fast peripheral port read data signals

Details of the AXI fast peripheral port read data signals.

Table A-16 AXI fast peripheral port read data signals

Name	Type	Source/destination	Description
RVALIDMFPx	Input	AXI3 fast peripheral port IO	Read data valid
RREADYMFPx	Output		Read data ready
RLASTMFPx	Input		Read data last
RDATAMFPx[31:0]	Input		Read data
RRESPMFPx[1:0]	Input		Read data response

AXI fast peripheral port write address signals

Details of the AXI fast peripheral port write address signals.

Table A-17 AXI fast peripheral port write address signals

Name	Type	Source/destination	Description
AWVALIDMFPx	Output	AXI3 fast peripheral port IO	Write address valid
AWREADYMFPx	Input		Write address ready
AWADDRMFPx[31:0]	Output		Write address
AWSIZEMFPx[1:0]	Output		Write address burst size
AWLENMFPx[3:0]	Output		Write address burst length
AWBURSTMFPx[1:0]	Output		Write address burst type
AWCACHEMFPx[3:0]	Output		Write address cache type
AWPROTMFPx[2:0]	Output		Write address protection type
AWLOCKMFPx[1:0]	Output		Write address lock type

AXI fast peripheral port write data signals

Details of the AXI fast peripheral port write data signals.

Table A-18 AXI fast peripheral port write data signals

Name	Type	Source/destination	Description
WVALIDMFPx	Output	AXI3 fast peripheral port IO	Write data valid
WREADYMFPx	Input		Write data ready
WLASTMFPx	Output		Write data last
WSTRBMFPx[3:0]	Output		Write data strobes
WDATAMFPx[31:0]	Output		Write data

AXI fast peripheral port write response signals

Details of the AXI fast peripheral port write response signals.

Table A-19 AXI fast peripheral port write response signals

Name	Type	Source/destination	Description
BVALIDMFPx	Input	AXI3 fast peripheral port IO	Write response valid
BREADYMFPx	Output		Write response ready
BRESPMFPx[1:0]	Input		Write response

A.8.3 AXI low-latency peripheral port signals

Cortex-R8 processor AXI low-latency peripheral port signals.

AXI low-latency peripheral clock enable signals

Details of the AXI low-latency peripheral port clock enable signals.

Table A-20 AXI low-latency peripheral clock enable signals

Name	Type	Source/destination	Description
INCLKENMP	Input	**CLK**	Clock enable
OUTCLKENMP	Input		Clock enable

AXI low-latency peripheral port read address signals

Details of the AXI low-latency peripheral port read address signals.

Table A-21 AXI low-latency peripheral port read address signals

Name	Type	Source/destination	Description
ARADDRMP[31:0]	Output	AXI3 low-latency peripheral port	Read address
ARBURSTMP[1:0]	Output		Read address burst type
ARCACHEMP[3:0]	Output		Read address cache type
ARIDMP[n][cq]	Output		Read address ID
ARLENMP[3:0]	Output		Read address burst length
ARLOCKMP[1:0]	Output		Read address lock type
ARPROTMP[2:0]	Output		Read address protection type
ARREADYMP	Input		Read address ready
ARSIZEMP[1:0]	Output		Read address burst size
ARUSERMP[9:0]	Output		Read address transfer attributes, see *12.1.2 AXI3 USER bits* on page 12-345
ARVALIDMP	Output		Read address valid

AXI low-latency peripheral port read data signals

Details of the AXI low-latency peripheral port read data signals.

Table A-22 AXI low-latency peripheral port read data signals

Name	Type	Source/destination	Description
RVALIDMP	Input	AXI3 low-latency peripheral port	Read data valid
RREADYMP	Output		Read data ready
RIDMP[n][cr]	Input		Read data ID
RLASTMP	Input		Read data last
RDATAMP[31:0]	Input		Read data
RRESPMP[1:0]	Input		Read data response

AXI low-latency peripheral port write address signals

Details of the AXI low-latency peripheral port write address signals.

[cq] 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 [5:0].

[cr] 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 [5:0].

Table A-23 AXI low-latency peripheral port write address signals

Name	Type	Source/destination	Description
AWVALIDMP	Output	AXI3 low-latency peripheral port	Write address valid
AWREADYMP	Input		Write address ready
AWIDMP[n][cs]	Output		Write address ID
AWADDRMP[31:0]	Output		Write address
AWSIZEMP[1:0]	Output		Write address burst size
AWLENMP[3:0]	Output		Write address burst length
AWBURSTMP[1:0]	Output		Write address burst type
AWCACHEMP[3:0]	Output		Write address cache type
AWPROTMP[2:0]	Output		Write address protection type
AWLOCKMP[1:0]	Output		Write address lock type
AWUSERMP[11:0]	Output		Write address transfer attributes, see *12.1.2 AXI3 USER bits* on page 12-345

AXI low-latency peripheral port write data signals

Details of the AXI low-latency peripheral port write data signals.

Table A-24 AXI low-latency peripheral port write data signals

Name	Type	Source/destination	Description
WVALIDMP	Output	AXI3 low-latency peripheral port	Write data valid
WREADYMP	Input		Write data ready
WIDMP[n][ct]	Output		Write data ID
WLASTMP	Output		Write data last
WSTRBMP[3:0]	Output		Write data strobes
WDATAMP[31:0]	Output		Write data
WUSERMP[2:0]	Output		Write data transfer attributes, see *12.1.2 AXI3 USER bits* on page 12-345

AXI low-latency peripheral port write response signals

Details of the AXI low-latency peripheral port write response signals.

[cs] 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 [5:0].

[ct] 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 [5:0].

Table A-25 AXI low-latency peripheral port write response signals

Name	Type	Source/destination	Description
BVALIDMP	Input	AXI3 low-latency peripheral port	Write response valid
BREADYMP	Output		Write response ready
BIDMP[n][cu]	Input		Write response ID
BRESPMP[1:0]	Input		Write response

A.8.4 AXI ACP slave port signals

Details of the AXI ACP slave port signals that are present on the Cortex-R8 processor.

AXI ACP slave port clock enable signal

Signal name, type, and source or destination information for the AXI ACP slave port clock enable signal.

Table A-26 AXI ACP slave port clock enable signal

Name	Type	Source/destination	Description
ACLKENSC	Input	Clock controller	Clock bus enable for the AXI bus that enables the AXI interface to operate at integer ratios of the system clock.

AXI ACP slave port read address signals

Signal type and source or destination information for the AXI ACP slave port read address signals.

Table A-27 Signal type and source or destination information for the XXX signals.

Name	Type	Source/destination	Description
ARVALIDSC	Input	AXI3 device	Address valid.
ARREADYSC	Output		Address ready.
ARIDSC[n]	Input		Address ID. You can define the number of AXI ID bits on this port using the `AXISC_ID_BIT` build parameter.
ARADDRSC[31:0]	Input		Address.
ARSIZESC[1:0]	Input		Burst size.
ARLENSC[3:0]	Input		Burst length.
ARBURSTSC[1:0]	Input		Burst type.
ARCACHESC[3:0]	Input		Cache type.
ARPROTSC[2:0]	Input		Protection type.
ARLOCKSC	Input		Lock type.
ARUSERSC[4:0]	Input		Transfer attributes.

AXI ACP slave port read data signals

Summary of the AXI ACP slave port read data signals.

[cu] 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 [5:0].

Table A-28 AXI ACP slave port read data signals

Name	Type	Source/destination	Description
RVALIDSC	Output	AXI3 device	Read valid.
RREADYSC	Input		Read ready.
RIDSC[n]	Output		Read ID. You can define the number of AXI ID bits on this port using the `AXISC_ID_BIT` build parameter.
RLASTSC	Output		Read last.
RDATASC[63:0]	Output		Read data.
RRESPSC[1:0]	Output		Read response.

AXI ACP slave port write address signals

Signal name, type, and source or destination information for the AXI ACP slave port write address signals.

Table A-29 AXI ACP slave port write address signals

Name	Type	Source/destination	Description
AWVALIDSC	Input	AXI3 device	Address valid.
AWREADYSC	Output		Address ready.
AWIDSC[n]	Input		Address ID. You can define the number of AXI ID bits on this port using the `AXISC_ID_BIT` build parameter.
AWADDRSC[31:0]	Input		Address.
AWSIZESC[1:0]	Input		Burst size.
AWLENSC[3:0]	Input		Burst length. The maximum burst transfer must correspond to an L1 cache line, that is, 256 bits.
AWBURSTSC[1:0]	Input		Burst type.
AWCACHESC[3:0]	Input		Cache type.
AWPROTSC[2:0]	Input		Protection type.
AWLOCKSC	Input		Lock type.
AWUSERSC[5:0]	Input		Transfer attributes.

AXI ACP slave port write data signals

Signal name, type, and source or destination information for the AXI ACP slave port write data signals.

Table A-30 AXI ACP slave port write data signals

Name	Type	Source/destination	Description
WVALIDSC	Input	AXI3 device	Write valid.
WREADYSC	Output		Write ready.
WIDSC[n]	Input		Write ID. You can define the number of AXI ID bits on this port using the `AXISC_ID_BIT` build parameter.
WLASTSC	Input	AXI3 device	Write last.
WSTRBSC[7:0]	Input		Write strobes.
WDATASC[63:0]	Input		Write data.

AXI ACP slave port write response signals

Signal name, type, and source or destination information for the AXI ACP slave port write response signals.

Table A-31 AXI ACP slave port write response signals

Name	Type	Source/destination	Description
BVALIDSC	Output	AXI3 device	Response valid.
BREADYSC	Input		Response ready.
BIDSC[n]	Output		Response ID. You can define the number of AXI ID bits on this port using the `AXISC_ID_BIT` build parameter.
BRESPSC[1:0]	Output		Write response.

A.8.5 AXI TCM slave port signals

Cortex-R8 processor AXI TCM slave port signals.

AXI TCM slave port clock enable signal

Details of the AXI TCM slave port clock enable signal.

Table A-32 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

AXI TCM slave port read address signals

Details of the AXI TCM slave port read address signals.

Table A-33 AXI TCM slave port read address signals

Name	Type	Source/destination	Description
ARVALIDST	Input	AXI3 device	Read address valid
ARREADYST	Output		Read address ready
ARIDST[n][cv]	Input		Read address ID
ARADDRST[31:0]	Input		Read address
ARSIZEST[1:0]	Input		Read address burst size
ARLENST[3:0]	Input		Read address burst length
ARBURSTST[1:0]	Input		Read address burst type
ARUSERST[2:0]	Input		Read address transfer attributes
ARCACHEST[3:0]	Input		Read address cache type
ARLOCKST[1:0]	Input		Read address lock type
ARPROTST[2:0]	Input		Read address protection type

AXI TCM slave port read data signals

Details of the AXI TCM slave port read data signals.

Table A-34 AXI TCM slave port read data signals

Name	Type	Source/destination	Description
RVALIDST	Output	AXI3 device	Read data valid
RREADYST	Input		Read data ready
RIDST[n][cw]	Output		Read data ID
RLASTST	Output		Read data last
RDATAST[63:0]	Output		Read data
RRESPST[1:0]	Output		Read data response

AXI TCM slave port write address signal

Details of the AXI TCM slave port write address signals.

[cv] You can define the number of AXI ID bits on this port using the `AXIST_ID_BIT` build parameter.
[cw] You can define the number of AXI ID bits on this port using the `AXIST_ID_BIT` build parameter.

Table A-35 AXI TCM slave port write address signals

Name	Type	Source/destination	Description
AWVALIDST	Input	AXI3 device	Write address valid.
AWREADYST	Output		Write address ready.
AWIDST[n][cx]	Input		Write address ID.
AWADDRST[31:0]	Input		Write address.
AWSIZEST[1:0]	Input		Write address burst size.
AWLENST[3:0]	Input		Write address burst length. The maximum burst transfer must correspond to an L1 cache line, that is, 256 bits.
AWBURSTST[1:0]	Input		Write address burst type.
AWUSERST[2:0]	Input		Write address transfer attributes.
AWCACHEST[3:0]	Input		Write address cache type.
AWLOCKST[1:0]	Input		Write address lock type.
AWPROTST[2:0]	Input		Write address protection type.

AXI TCM slave port write data signals

Details of the AXI TCM slave port write data signals.

Table A-36 AXI TCM slave port write data signals

Name	Type	Source/destination	Description
WVALIDST	Input	AXI3 device	Write data valid
WREADYST	Output		Write data ready
WIDST[n][cy]	Input		Write data ID
WLASTST	Input		Write data last
WSTRBST[7:0]	Input		Write data strobes
WDATAST[63:0]	Input		Write data

AXI TCM slave port write response signals

Details of the AXI TCM slave port write response signals.

Table A-37 AXI TCM slave port write response signals

Name	Type	Source/destination	Description
BVALIDST	Output	AXI3 device	Write response valid
BREADYST	Input		Write response ready
BIDST[n][cz]	Output		Write response ID
BRESPST[1:0]	Output		Write response

[cx] You can define the number of AXI ID bits on this port using the `AXIST_ID_BIT` build parameter.
[cy] You can define the number of AXI ID bits on this port using the `AXIST_ID_BIT` build parameter.
[cz] You can define the number of AXI ID bits on this port using the `AXIST_ID_BIT` build parameter.

A.9 Performance monitoring signals

Details of the Cortex-R8 processor performance monitoring signals.

Table A-38 Performance monitoring signals

Name	Type	Source/destination	Description
PMUEVENTx[55:0]	Output	*Performance Monitoring Unit* (PMU) or External Performance Monitoring Unit	PMU event bus for the corresponding core.
PMUIRQ[CN:0]	Output	System Integrity Controller or External Performance Monitoring unit	Core interrupt request by system metrics.
PMUPRIV[CN:0]	Output	External Performance Monitoring Unit	Gives the status of the core: `0b0` In user mode. `0b1` In privileged mode. **Note** This signal does not provide input to the CoreSight trace delivery infrastructure.

Related reference

10.1 Performance Monitoring Unit on page 10-215

A.10 Exception flag signals

Details of the Cortex-R8 processor exception flag signals.

Table A-39 Exception flag signals

Name	Type	Source/destination	Description
SCUEVABORT	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 **CLK** clock cycle.
FPUFLAGSx[5:0]	Output		Floating-Point Unit output flags for the corresponding core. Only implemented if the corresponding core includes an FPU: Bit[5] gives the value of FPSCR[7]. Bits[4:0] give the value of FPSCR[4:0].

Related reference

Chapter 5 Floating Point Unit Programmers Model on page 5-99

A.11 Error detection notification signals

Cortex-R8 processor error detection notification signals.

This section contains the following subsections:

A.11.1 Error detection global notification signals

Details of the error detection global notification signals.

Table A-40 Error detection global notification signals

Name	Type	Source/destination	Description
RAMERR	Output	Core 0-3 RAM arrays and SCU RAMs	Any ECC error on any RAM
FATALRAMERR[CN:0]	Output		Fatal ECC error on any RAM
ITCMECCEN	Input		Defines reset value of ACTLR bit[10] for each core
FATALERRDET	Output		Fatal ECC error on any core
CORRBUSERR	Output	SCU	Correctable ECC error on any bus
FATALBUSERR	Output		Fatal ECC error on any bus

A.11.2 RAM ECC error bank status signals

Details of the RAM ECC error bank status signals.

Table A-41 RAM ECC error bank status signals

Name	Type	Source/destination	Description
DCEBEMPTY[CN:0]	Output	Specific RAM group	ECC error bank empty for data cache
ICEBEMPTY[CN:0]	Output		ECC error bank empty for instruction cache
DTCMEBEMPTYx	Output		ECC error bank empty for data TCM for the corresponding core
ITCMEBEMPTYx	Output		ECC error bank empty for instruction TCM for the corresponding core
SCUEBEMPTY	Output		ECC error bank empty for SCU

A.11.3 Bus ECC error signals on AXI master ports

Details of the Bus ECC error signals on AXI master ports. These signals are only present if ECC is implemented. The **x** at the end of the signal name represents either **0** for AXI master port 0 or **1** for the optional AXI master port 1.

Table A-42 Bus ECC error signals on AXI master ports

Name	Type	Source/destination	Description
AXICORRERRMx	Output	AXI master port	Correctable error on DR channel of AXI master port.
AXIFATALERRMx[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.
ARVALIDPTYMx	Output		Parity for address valid.
ARREADYPTYMx	Input		Parity for address ready.
ARADDRPTYMx[3:0]	Output		Parity for address.
ARCTLPTYMx[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.
ARUSERPTYMx	Output		Parity for transfer attributes.
RVALIDPTYMx	Input		Parity for read valid.
RREADYPTYMx	Output		Parity for read ready.
RCTLPTYMx[1:0]	Input		Parity signals: [0] parity for read ID. [1] parity for read response and read last.
RDATAERRCODEMx[7:0]	Input		ECC bits on data bus, when `BUS_ECC` build parameter is set.
AWVALIDPTYMx	Output		Parity for address valid.
AWREADYPTYMx	Input		Parity for address ready.
AWADDRPTYMx[3:0]	Output		Parity for address.
AWCTLPTYMx[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.
AWUSERPTYMx	Output		Parity for transfer attributes.
WVALIDPTYMx	Output		Parity for write valid.
WREADYPTYMx	Input		Parity for write ready.

Table A-42 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 fast peripheral port

Details of the bus ECC error signals on the AXI fast peripheral port. These signals are only present if ECC is implemented.

Table A-43 Bus ECC error signals on AXI fast peripheral port

Name	Type	Source/destination	Description
AXICORRERRMFPx	Output	AXI fast peripheral port IO	Correctable error on DR channel of AXI fast peripheral port.
AXIFATALERRMFPx[4:0]	Output		Fatal error on AXI fast peripheral 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.
ARVALIDPTYMFPx	Output		Parity for address valid.
ARREADYPTYMFPx	Input		Parity for address ready.
ARADDRPTYMFPx[3:0]	Output		Parity for address.
ARCTLPTYMFPx[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.
RVALIDPTYMFPx	Input		Parity for read valid.
RREADYPTYMFPx	Output		Parity for read ready.
RCTLPTYMFPx[1:0]	Input		Parity signals: [0] - [1] parity for read response.

Table A-43 Bus ECC error signals on AXI fast peripheral port (continued)

Name	Type	Source/destination	Description
RDATAERRCODEMFPx[6:0]	Input	AXI fast peripheral port IO	ECC bits on data bus, when `BUS_ECC` build parameter is set.
AWVALIDPTYMFPx	Output		Parity for address valid.
AWREADYPTYMFPx	Input		Parity for address ready.
AWADDRPTYMFPx[3:0]	Output		Parity for address.
AWCTLPTYMFPx[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.
WVALIDPTYMFPx	Output		Parity for write valid.
WREADYPTYMFPx	Input		Parity for write ready.
WCTLPTYMFPx[2:0]	Output		Parity signals: [0] parity for write ID. [1] parity for write strobes. [2] parity for write last.
WDATAERRCODEMFPx[6:0]	Output		ECC bits on data bus, when `BUS_ECC` build parameter is set.
BVALIDPTYMFPx	Input		Parity for response valid.
BREADYPTYMFPx	Output		Parity for response ready.
BCTLPTYMFPx[1:0]	Input		Parity signals: [0] parity for response ID. [1] parity for write response.

A.11.5 Bus ECC error signals on AXI low-latency peripheral port

Details of the bus ECC error signals on the AXI low-latency peripheral port. These signals are only present if ECC is implemented.

Table A-44 Bus ECC error signals on AXI low-latency peripheral port

Name	Type	Source/destination	Description
AXICORRERRMP	Output	AXI low-latency peripheral port	Correctable error on DR channel of AXI low-latency peripheral port.
AXIFATALERRMP[4:0]	Output		Fatal error on AXI low-latency peripheral 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.
RDATAERRCODEMP[6:0]	Input		ECC bits on data bus, when `BUS_ECC` build parameter is set.
AWVALIDPTYMP	Output		Parity for address valid.
AWREADYPTYMP	Input		Parity for address ready.
AWADDRPTYMP[3:0]	Output		Parity for address.
AWCTLPTYMP[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.
AWUSERPTYMP	Output		Parity for transfer attributes.
WVALIDPTYMP	Output		Parity for write valid.
WREADYPTYMP	Input		Parity for write ready.

Table A-44 Bus ECC error signals on AXI low-latency peripheral port (continued)

Name	Type	Source/destination	Description
WCTLPTYMP[2:0]	Output	AXI low-latency peripheral port	Parity signals: [0] parity for write ID. [1] parity for write strobes. [2] parity for write last.
WUSERPTYMP	Output		Parity for transfer attributes.
WDATAERRCODEMP[6:0]	Output		ECC bits on data bus, when `BUS_ECC` build parameter is set.
BVALIDPTYMP	Input		Parity for response valid.
BREADYPTYMP	Output		Parity for response ready.
BCTLPTYMP[1:0]	Input		Parity signals: [0] parity for response ID. [1] parity for write response.

A.11.6 Bus ECC error signals on AXI ACP slave port

Details of the bus ECC error signals on the optional AXI ACP slave port. These signals are only present if ECC is implemented.

Table A-45 Bus ECC error signals on AXI ACP slave port

Name	Type	Source/destination	Description
AXICORRERRSC	Output	AXI ACP	Correctable error on DW channel of AXI ACP.
AXIFATALERRSC[4:0]	Output		Fatal error on AXI ACP: [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.
ARVALIDPTYSC	Input		Parity for address valid.
ARREADYPTYSC	Output		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.

Table A-45 Bus ECC error signals on AXI ACP slave port (continued)

Name	Type	Source/destination	Description
RDATAERRCODESC[7:0]	Output	AXI ACP	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.11.7 Bus ECC error signals on AXI TCM slave port

Details of the bus ECC error signals on the AXI TCM slave port. These signals are only present if ECC is implemented.

Table A-46 Bus ECC error signals on AXI TCM slave port

Name	Type	Source/destination	Description
AXICORRERRST	Output	AXI TCM slave port	Correctable error on DW channel of AXI TCM slave port.
AXIFATALERRST[4:0]	Output		Fatal error on AXI TCM slave 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.
ARVALIDPTYST	Input		Parity for read address valid.
ARREADYPTYST	Output		Parity for read address ready.
ARADDRPTYST[3:0]	Input		Parity for read address.
ARCTLPTYST[3:0]	Input		Parity signals: [0] parity for read address ID. [1] parity for read address burst length. [2] parity for read address burst size, burst type, and lock type. [3] parity for read address cache type and protection.
ARUSERPTYST	Input		Parity for transfer attributes.
RVALIDPTYST	Output		Parity for read valid.
RREADYPTYST	Input		Parity for read ready.
RCTLPTYST[1:0]	Output		Parity signals: [0] parity for read response, read last. [1] parity for read ID.
RDATAERRCODEST[7:0]	Output		ECC bits on data bus, when `BUS_ECC` build parameter is set.
AWVALIDPTYST	Input		Parity for write address valid.
AWREADYPTYST	Output		Parity for write address ready.
AWADDRPTYST[3:0]	Input		Parity for write address.
AWCTLPTYST[3:0]	Input		Parity signals: [0] parity for write ID. [1] parity for write burst length. [2] parity for write burst size, burst type, and lock type. [3] parity for write cache type and protection.
AWUSERPTYST	Input		Parity for transfer attributes.
WVALIDPTYST	Input		Parity for write valid.
WREADYPTYST	Output		Parity for write ready.
WCTLPTYST[2:0]	Input		Parity signals: [0] parity for write ID. [1] parity for write strobes
WDATAERRCODEST[7:0]	Input		ECC bits on data bus, when `BUS_ECC` build parameter is set.

Table A-46 Bus ECC error signals on AXI TCM slave port (continued)

Name	Type	Source/destination	Description
BVALIDPTYST	Output	AXI TCM slave port	Parity for write response valid.
BREADYPTYST	Input		Parity for write response ready.
BCTLPTYST[1:0]	Output		Parity signals: [0] parity for write response ID. [1] parity for write response.

A.11.8 Lock-step and split/lock signals

Details of the lock-step and split/lock signals.

Table A-47 Lock-step and split/lock signals

Name	Type	Source/destination	Description
COMPENABLE	Input	System configuration	Enables comparison logic that compares the outputs of the core, SCU, and AXI TCM slave with those of their redundant copy. See *1.6 Redundant processor core comparison* on page 1-20.
COMPFAULT	Output		Indicates a fault in either the main or redundant logic. See *1.6 Redundant processor core comparison* on page 1-20.
SAFEMODE	Input		Selects split/lock mode or lock-step mode. See *1.6.1 Split/lock* on page 1-20.

A.12 Test interface

Details of the test interface signals.

Table A-48 Test interface signals

Name	Type	Source/destination	Description
DFTSE	Input	External test interface	Scan shift enable
DFTRAMHOLD	Input		Holds RAM content during scan shift
DFTRAMCLKENABLE	Input		Forces RAM clock for DFT purposes even when cores are in WFI mode
DFTTESTMODE	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.

A.13.1 L1 and DTCM Cortex®-R8 processor MBIST interface width without ECC

Details of the L1 and DTCM signals for designs without ECC.

Note

All MBIST signals except **MBISTREQx** are internal pins of `PLOVER.v` corresponding to `plover_mbist_intf<x>.v` modules. The MBIST controller must connect directly to the `plover_mbist_intf<x>.v` module.

Table A-49 L1 and DTCM Cortex-R8 processor MBIST interface width without ECC

Name	Type	Source/destination	Description
MBISTREQ1	Input	MBIST controller	BIST mode request signal, one per core

A.13.2 L1 and DTCM Cortex®-R8 processor MBIST interface width with ECC

Details of the the L1 and DTCM signals for designs with ECC.

Table A-50 L1 and DTCM Cortex-R8 processor MBIST interface width with ECC

Name	Type	Source/destination	Description
MBISTREQ1[1:0]	Input	MBIST controller	BIST mode request signal, one per core

A.13.3 ITCM Cortex®-R8 processor MBIST interface width without ECC

Details of the ITCM signals for designs without ECC.

Table A-51 ITCM Cortex-R8 processor MBIST interface width without ECC

Name	Type	Source/destination	Description
MBISTREQ2	Input	MBIST controller	BIST mode request signal

A.13.4 ITCM Cortex®-R8 processor MBIST interface width with ECC

Details of the ITCM signals for designs with ECC.

Table A-52 ITCM Cortex-R8 processor MBIST interface width with ECC

Name	Type	Source/destination	Description
MBISTREQ2[1:0]	Input	MBIST controller	BIST mode request signal

A.14 External debug signals

Details of the Cortex-R8 processor external debug signals.

This section contains the following subsections:

A.14.1 Debug enable signals

Details of the debug enable signals.

Table A-53 Debug enable signals

Name	Type	Source/destination	Description
DBGEN[CN:0]	Input	External debug device	Individual core invasive debug enable: `0b0` Not enabled. `0b1` Enabled.
NIDEN[CN:0]	Input		Individual core non-invasive debug enable: `0b0` Not enabled. `0b1` Enabled.

A.14.2 Debug signals

Details of the debug signals.

Table A-54 Debug signals

Name	Type	Source/ destination	Description
EDBGRQ[CN:0]	Input	External debug device	Individual core external debug request: `0b0` No external debug request. `0b1` External debug request. The core treats the **EDBGRQ** input as level sensitive. The **EDBGRQ** input must be asserted until the core asserts **DBGACK**.
DBGACK[CN:0]	Output		Individual core debug acknowledge signal. Acknowledges that the corresponding core has entered debug state after an external debug request.
DBGCPUDONE[CN:0]	Output		Acknowledges that corresponding core has entered debug state and that all previous non-debug state memory accesses are complete.
DBGNOPWRDWN[CN:0]	Output		Output reflecting the value of DBGPRCR[0]. See the *Arm® Architecture Reference Manual Arm®v7-A and Arm®v7-R edition*.
DBGSWENABLE[CN:0]	Input		When LOW only the external debug agent can modify debug registers: `0b0` Not enabled. `0b1` Enabled. Access by the software through the extended CP14 interface is permitted. External CP14 and external debug accesses are permitted.
DBGROMADDR[31:12]	Input	External debug device	CoreSight System configuration. Specifies bits[31:12] of the ROM table physical address. If the address cannot be determined, tie off this signal to zero.
DBGROMADDRV	Input		Valid signal for **DBGROMADDR**. If the address cannot be determined, tie this signal LOW.
DBGSELFADDR[31:17]	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 off this signal to zero.
DBGSELFADDRV	Input		Valid signal for **DBGSELFADDR**. If the offset cannot be determined, tie this signal LOW.

A.14.3 Miscellaneous debug signals

Details of the miscellaneous debug signals.

Table A-55 Miscellaneous debug signals

Name	Type	Source/destination	Description
COMMRX[CN:0]	Output	External debug device	Individual core signal. Comms Channels Receive portion of Data Transfer Register full flag: `0b0` Empty. `0b1` Full.
COMMTX[CN:0]	Output		Individual core signal. Comms Channels Transmit portion of Data Transfer Register empty flag: `0b0` Full. `0b1` Empty.

A.14.4 Debug APB interface signals

Details of the Debug APB interface signals.

Table A-56 Debug APB interface signals

Name	Type	Source/destination	Description
PENABLEDBG	Input	CoreSight APB devices	APB clock enable. Indicates a second and subsequent cycle of a transfer.
PRDATADBG[31:0]	Output		APB read data bus.
PSELDBG	Input		Debug registers select: `0b0` Debug registers not selected. `0b1` Debug registers selected.
PSLVERRDBG	Output		APB slave error signal: `0b0` No transfer error. `0b1` Transfer error.
PWRITEDBG	Input		APB read/write signal.

Table A-56 Debug APB interface signals (continued)

Name	Type	Source/destination	Description
PADDRDBG[16:2]	Input	CoreSight APB devices	Programming address. Bits[16:12] have the following meaning: `0b00000` ROM table. `0b10000` Core 0 debug. `0b10001` Core 0 PMU. `0b10010` Core 1 debug, if core 1 is present, otherwise reserved. `0b10011` Core 1 PMU, if core 1 is present, otherwise reserved. `0b10100` Core 2 debug, if core 2 is present, otherwise reserved. `0b10101` Core 2 PMU, if core 2 is present, otherwise reserved. `0b10110` Core 3 debug, if core 3 is present, otherwise reserved. `0b10111` Core 3 PMU, if core 3 is present, otherwise reserved. `0b11000` CTI0. `0b11001` CTI1, if core 1 is present, otherwise reserved. `0b11010` CTI2, if core 2 is present, otherwise reserved. `0b11011` CTI3, if core 3 is present, otherwise reserved. `0b11100` ETM0. `0b11101` ETM1, if ETM1 is present, otherwise reserved. `0b11110` ETM2, if ETM2 is present, otherwise reserved. `0b11111` ETM3, if ETM3 is present, otherwise reserved.
PADDRDBG31	Input		APB address bus bit[31]: `0b0` Not an external debugger access. `0b1` External debugger access.
PREADYDBG	Output		APB slave ready. An APB slave can assert **PREADY** to extend a transfer.
PWDATADBG[31:0]	Input		APB write data.

A.15 ETM signals

Details of the ETM processor trace interface signals from the Cortex-R8 processor. The **x** at the end of the signal name represents **0** for ETM0, **1** for ETM1, **2** for ETM2, or **3** for ETM3, if implemented.

This section contains the following subsections:

A.15.1 Processor trace interface signals

Details of the processor trace interface signals.

Table A-57 Processor trace interface signals

Signal name	Type	Source/destination	Description
ETMBUS[321:0]	Input	Processor	Combined ETM interface channel
ETMIFVALID	Input		Core active, interface stable
ETMIFENx	Output		Power control for ETM processor trace interface
ETMBACK	Output		Configurable output to stall processor
DBGACK[CN:0]	Input		Core is in debug state
CPUACTIVE	Input		Core is not in WFI/WFE or other low-power state

A.15.2 ETM APB signals

Details of the ETM APB signals.

Table A-58 ETM APB signals

Signal name	Type	Source/destination	Description
PADDRDBG[16:2]	Input	Debug APB interconnect	Debug APB Address Bus.
PADDRDBG31	Input		Originates as an output signal from the *Debug Access Port* (DAP): `0b0` Indicates an access from software. `0b1` Indicates an access from hardware (JTAG).
PENABLEDBG	Input		The Debug APB interface is enabled for a transfer.
PSELDBG	Input		Debug APB slave select signal.
PREADYDBG	Output		Extends Debug APB transfers.
PRDATADBG[31:0]	Output		Debug APB read data.
PWDATADBG[31:0]	Input		Debug APB write data.
PWRITEDBG	Input		Debug APB transfer direction: `0b0` = Read, `0b1` = Write.
PSLVERRDBG	Output		Debug APB error response.

A.15.3 ETM ATB signals for instruction trace

Details of the ETM ATB signals for instruction trace.

Table A-59 ETM ATB signals for instruction trace

Signal name	Type	Source/destination	Description
AFREADYMIx	Output	CoreSight trace system	ATB interface FIFO flush finished
AFVALIDMIx	Input		ATB interface FIFO flush request
ATBYTESMIx[1:0]	Output		Size of **ATDATA**
ATDATAMIx[31:0]	Output		ATB interface data
ATIDMIx[6:0]	Output		ATB interface trace source ID
ATREADYMIx	Input		**ATDATA** can be accepted
ATVALIDMIx	Output		ATB interface data valid
SYNCREQIx	Input		Synchronization request from instruction trace sink

A.15.4 ETM ATB signals for data trace

Details of the ETM ATB signals for data trace.

Table A-60 ETM ATB signals for data trace

Signal name	Type	Source/destination	Description
AFREADYMDx	Output	CoreSight trace system	ATB interface FIFO flush finished
AFVALIDMDx	Input		ATB interface FIFO flush request
ATBYTESMDx[2:0]	Output		Size of **ATDATA**
ATDATAMDx[63:0]	Output		ATB interface data
ATIDMDx[6:0]	Output		ATB interface trace source ID
ATREADYMDx	Input		**ATDATA** can be accepted
ATVALIDMDx	Output		ATB interface data valid
SYNCREQDx	Input		Synchronization request from data trace sink

A.15.5 ETM Miscellaneous signals

Details of the ETM miscellaneous signals.

Table A-61 ETM Miscellaneous signals

Signal name	Type	Source/destination	Description
PROCSEL[2:0]	Output	Trace multiplexor, if present	Where an ETM is shared between multiple processors, this signal controls the multiplexor. The value is driven from bits[2:0] of the *11.8.2 Processor Select Control Register* on page 11-268.
NUMPROC[2:0]	Input	Tie off	Where an ETM is shared between multiple cores, this signal specifies the number of cores the ETM can trace. It must be tied to the number of cores sharing the ETM minus 1. These signals determine the value of bits[30:28] in the *11.8.33 ID Register 3* on page 11-302.
NIDEN[CN:0]	Input	System	Non-invasive debug enable. When HIGH (`0b1`), indicates that non-invasive debug is enabled.
ETMACTIVEx	Output	Processor	Trace is being output.
ETMEVENT[63:0]	Input	PMU and CTI	External input resources.
ETMEXTOUT[3:0]	Output	CTI	External outputs.
SYSSTALL	Input	Tie off	System supports stalling of the core by the ETM.
ETMPWRUPREQx	Output	System power control	Request to maintain power to ETM.
TSSIZE	Input	Tie off	When HIGH (`0b1`), timestamp is 64 bits. When LOW (`0b0`), timestamp is 48 bits.
TSVALUE[63:0]	Input	CoreSight system	Timestamp value.
CLUSTERID[3:0]	Input	System	Value read in the Cluster ID field, bits[11:8], of the Cortex-R8 *Multiprocessor Affinity Register* (MPIDR).
CPUID	Input	System	Value read in the CPU ID field, bits[1:0], of the MPIDR in the connected core.

A.15.6 CTI signals

Details of the CTI signals.

Table A-62 CTI signals

Signal name	Type	Source/destination	Description
CTICHIN[3:0]	Input	Trace device	Channel in
CTICHOUTACK[3:0]	Input		Channel out acknowledge
CTICHOUT[3:0]	Output		Channel out
CTICHINACK[3:0]	Output		Channel in acknowledge
CIHSBYPASS[3:0]	Input		Channel interface HS bypass
nCTIIRQ[CN:0]	Output		Active-LOW interrupt from CTI

A.16 Memory reconstruction port signals

Details of the Cortex-R8 processor MRP signals. The **x** at the end of the signal name represents either core **0**, core **1**, core **2** or core 3.

Table A-63 Memory reconstruction port signals

Name	Type	Source/destination	Description
MRPREADYx	Input	Trace analysis engine	Ready signal of any write access
MRPVALIDx	Output		Valid signal of any write access
MRPADDRx[31:0]	Output		Address of any write access
MRPDATAx[63:0]	Output		Data of any write access
MRPSTRBx[7:0]	Output		Strobe of any write access

A.17 Power gating interface signals

Details of the Cortex-R8 processor power gating interface signals. The **x** at the end of the signal name represents either core **0**, core **1**, core **2** or core 3.

Table A-64 Power gating interface signals

Name	Type	Source/destination	Description
nPWRUPSCURAM	Input	Power controller	SCU power up switch enable
nPWRUPACKSCURAM	Output		SCU power up switch acknowledge
nISOLATESCURAM	Input		SCU RAM clamp control
nPWRUPACKCPUx_TRICKLE	Output		Individual core power up acknowledge
nPWRUPACKCPUx_HAMMER	Output		Individual core power up acknowledge
nPWRUPCPUx_TRICKLE	Input		Individual core power up request
nPWRUPCPUx_HAMMER	Input		Individual core power up request
nPWRUPCPUDRAMx	Input		Individual core data RAM power up switch enable
nPWRUPACKCPUDRAMx	Output		Individual core data RAM power up switch acknowledge
nRETCPUDRAMx	Input		Individual core data RAM retention control
nPWRUPCPUIRAMx	Input		Individual core instruction RAM power up switch enable
nPWRUPACKCPUIRAMx	Output		Individual core instruction RAM power up switch acknowledge
nRETCPUIRAMx	Input		Individual core instruction RAM retention control
nPWRUPCPUDTCMx	Input		Individual core DTCM RAM power up request
nPWRUPACKCPUDTCMx	Output		Individual core DTCM RAM power up acknowledge
nRETCPUDTCMx	Input		Individual core DTCM RAM retention control
nISOLATEDTCMx	Input		Individual DTCM RAM clamp control
nPWRUPCPUITCMx	Input		Individual core ITCM RAM power up request
nPWRUPACKCPUITCMx	Output		Individual core ITCM RAM power up acknowledge
nRETCPUITCMx	Input		Individual core ITCM RAM retention control
nISOLATEITCMx	Input		Individual ITCM RAM clamp control
nPWRUPDBG_TRICKLE	Input		Debug power up request
nPWRUPDBG_HAMMER	Input		Debug power up request

Table A-64 Power gating interface signals (continued)

Name	Type	Source/destination	Description
nPWRUPACKDBG_TRICKLE	Output	Power controller	Debug power up acknowledge
nPWRUPACKDBG_HAMMER	Output		Debug power up acknowledge
nISOLATEDBG	Input		Debug clamp control
nPWRUPETMx_TRICKLE	Input		Individual ETM power up request
nPWRUPETMx_HAMMER	Input		Individual ETM power up request
nPWRUPACKETMx_TRICKLE	Output		Individual ETM power up acknowledge
nPWRUPACKETMx_HAMMER	Output		Individual ETM power up acknowledge
nISOLATECPUx	Input		Individual core clamp control
nISOLATECPUDRAMx	Input		Individual core data RAM clamp control
nISOLATECPUIRAMx	Input		Individual core instruction RAM clamp control
nISOLATEETMx	Input		Individual ETM clamp control

Appendix B
Cycle Timings and Interlock Behavior

This appendix describes the cycle timings of integer instructions on Cortex-R8 processor cores.

It contains the following sections:

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-R8 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

Details of the execution unit cycle time for data-processing instructions.

The data-processing instructions cycle timings table 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 *B.6 Serializing instructions* on page Appx-B-416.	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-R8 processor has an optimized path from a load instruction to a subsequent data processing instruction, saving one 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.

This section contains the following subsections:

- *B.3.1 Single load and store operation cycle timings* on page Appx-B-410.
- *B.3.2 Load multiple operations cycle timings* on page Appx-B-411.
- *B.3.3 Store multiple operations cycle timings* on page Appx-B-412.

B.3.1 Single load and store operation cycle timings

Details of 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]`
`LDR, [reg imm]`
`LDR, [reg reg]`
`LDR, [reg reg LSL #2]`
`LDR, [reg reg LSL #3]`
	1	2	3

`LDR, [reg reg LSL reg]`
`LDR, [reg reg LSR reg]`
`LDR, [reg reg ASR reg]`
`LDR, [reg reg ROR reg]`
`LDR, [reg reg, RRX]`
	2	3	4

Table B-2 Single load and store operation cycle timings (continued)

Instruction cycles	AGU cycles	Result latency	
		Fast forward cases	Other cases

`LDRB, [reg]`
`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]`
	1	2	3

`LDRB, [reg reg LSL reg]`
`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]`
	2	3	4

B.3.2 Load multiple operations cycle timings

The Cortex-R8 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. in the following table 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}
`LDRD`
`RFE`
	1	2	2	3

`LDM`, {3 registers}	2	2	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, {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
LDM, {14 registers}	7	8	2	3
LDM, {15 registers}	8	8	2	3
LDM, {16 registers}	8	9	2	3

B.3.3 Store multiple operations cycle timings

Details of 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} STRD SRS	1	2
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

Table B-4 Store multiple operations cycle timings (continued)

Instruction	AGU cycles	
	Aligned on a 64-bit boundary	
	Yes	No
`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

Details of 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, SMMLA, SMMLAR, SMMLS, SMMLSR	2	4
SMLALD, SMLALDX, SMLSLD, SMLDLDX	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 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.
- Load instructions to the PC register are processed in the execution units as standard instructions.

Related concepts

8.3 Branch prediction on page 8-157

Related reference

B.2 Data-processing instructions on page Appx-B-409

B.3 Load and store instructions on page Appx-B-410

B.6 Serializing instructions

Out-of-order execution is not always possible. Some instructions are serializing. Serializing instructions force the Cortex-R8 processor to complete all modifications to flags and general-purpose registers by previous instructions before the next instruction is executed.

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.

It contains the following section:

- *C.1 Revisions* on page Appx-C-418.

C.1 Revisions

This appendix describes the technical changes between released issues of this book.

Table C-1 Issue 0000-01

Change	Location	Affects
First release for r0p1	-	-

Table C-2 Differences between Issue 0000-01 and Issue 0001-02

Change	Location	Affects
Second release for r0p1	-	-

Table C-3 Differences between Issue 0001-02 and Issue 0001-03

Change	Location	Affects
Tables added	*2.5.3 AXI low-latency peripheral port* on page 2-43 *2.5.4 AXI Fast Peripheral Port* on page 2-44 *2.5.5 AXI TCM slave port* on page 2-45 *2.5.6 AXI slave Accelerator Coherency Port* on page 2-48	r0p1
Note added	*4.3.14 ITCM Region Register* on page 4-89	r0p1
Bit details added.	*9.3.2 SCU Configuration Register* on page 9-171	r0p1
Footnotes updated	*A.8.1 AXI master interface signals* on page Appx-A-368 *A.8.3 AXI low-latency peripheral port signals* on page Appx-A-374	r0p1
Bit values changed	*A.11.4 Bus ECC error signals on AXI fast peripheral port* on page Appx-A-386	r0p1

Table C-4 Differences between Issue 0001-03 and Issue 0002-00

Change	Location	Affects
First release for r0p2	• Revisions history table • This Revisions appendix • MIDR reset value in *4.2 Register summary* on page 4-59, *4.2.1 c0 registers* on page 4-59, and *4.2.9 System identification, control, and configuration register* on page 4-64	r0p2
Added information on ITCM size limit	*8.6 Instruction and data TCM* on page 8-161	All versions
Added information on input synchronization	*2.3.3 Input synchronization* on page 2-33	All versions
Clarified CTDOR bit assignments	• *Using the CTDOR* on page 4-93 • *4.3.16 Cache and TCM Debug Operation Register* on page 4-91	All versions
Clarified FIQ description	• *9.4 Interrupt controller* on page 9-189 • *9.4.2 Interrupt distributor interrupt sources* on page 9-189	All versions
Clarified transfers for AXI	*12.1.1 Supported AXI3 transfers* on page 12-344	All versions

Table C-4 Differences between Issue 0001-03 and Issue 0002-00 (continued)

Change	Location	Affects
Separated power up and power down sequences	*Dormant mode* on page 2-38	All versions
Fixed incorrect offsets and registers for processor ID registers	*Processor ID Registers* on page 10-216	All versions
Added missing ACP signals	*A.8.4 AXI ACP slave port signals* on page Appx-A-377	All versions
Fixed ICDIIDR reset value	• *Distributor Implementer Identification Register* on page 9-194 • *Distributor register summary table* on page 9-191	All versions
Added powerdown sequence	*Powerdown sequence* on page 2-40	All versions
Made various editorial improvements	Throughout the document	All versions

Table C-5 Differences between Issue 0002-00 and Issue 0003-01

Change	Location	Affects
First release for r0p3.	-	-
Updated 9.4.2 Interrupt distributor interrupt sources.	*9.4.2 Interrupt distributor interrupt sources* on page 9-189	r0p3
Updated DBGCPUDONE section.	*DBGCPUDONE* on page 10-235	r0p3
Added Limitations of the core and AXI slave port interactions paragraph.	*2.5.5 AXI TCM slave port* on page 2-45	r0p3
Updated Preloading TCMs with ECC in 2.3.4 Initialization	*Preloading TCMs with ECC* on page 2-34	r0p3
Updated Asynchronous aborts in 6.2.1 Fault classes	*Asynchronous aborts* on page 6-113	r0p3