Cortex™-A9

Revision: r3p0

Technical Reference Manual

ARM DDI 0388G (ID072711)

Cortex-A9

Technical Reference Manual

Release Information

The following changes have been made to this book.

Change history

Date	Issue	Confidentiality	Change
31 March 2008	A	Non-Confidential	First release for r0p0
08 July 2008	B	Non-Confidential Restricted Access	First release for r0p1
17 December 2008	C	Non-Confidential Restricted Access	First release for r1p0
30 September 2009	D	Non-Confidential Restricted Access	First release for r2p0
27 November 2009	E	Non-Confidential	Second release for r2p0
30 April 2010	F	Non-Confidential	First release for r2p2
19 July 2011	G	Non-Confidential	First release for r3p0

Proprietary Notice

Words and logos marked with ® or ™ are registered trademarks or trademarks of ARM® in the EU and other countries, except as otherwise stated below in this proprietary notice. Other brands and names mentioned herein may be the trademarks of their respective owners.

Neither the whole nor any part of the information contained in, or the product described in, this document may be adapted or reproduced in any material form except with the prior written permission of the copyright holder.

The product described in this document is subject to continuous developments and improvements. All particulars of the product and its use contained in this document are given by ARM in good faith. However, all warranties implied or expressed, including but not limited to implied warranties of merchantability, or fitness for purpose, are excluded.

This document is intended only to assist the reader in the use of the product. ARM shall not be liable for any loss or damage arising from the use of any information in this document, or any error or omission in such information, or any incorrect use of the product.

Where the term ARM is used it means “ARM or any of its subsidiaries as appropriate”.

Confidentiality Status

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

Product Status

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

Web Address

`http://www.arm.com`

Contents
Cortex-A9 Technical Reference Manual

Chapter 4 System Control

Chapter 5 Jazelle DBX registers

Chapter 6 Memory Management Unit

Chapter 7 Level 1 Memory System

Chapter 8 Level 2 Memory Interface

Chapter 9 Preload Engine

Chapter 10 Debug

Chapter 11 Performance Monitoring Unit

Appendix A Signal Descriptions

Preface

This preface introduces the *Cortex-A9 Technical Reference Manual (TRM)*. It contains the following sections:

- *About this book* on page vii
- *Feedback* on page xi.

About this book

This book is for the Cortex-A9 processor.

Product revision status

The r*n*p*n* identifier indicates the revision status of the product described in this book, where:

r*n*	Identifies the major revision of the product.
p*n*	Identifies the minor revision or modification status of the product.

Intended audience

This book is written for hardware and software engineers implementing Cortex-A9 system designs. It provides information that enables designers to integrate the processor into a target system.

Note

- The Cortex-A9 processor is a single core processor.
- The multiprocessor variant, the Cortex-A9 MPCore™ processor, consists of between one and four Cortex-A9 processors and a *Snoop Control Unit* (SCU). See the *Cortex-A9 MPCore Technical Reference Manual* for a description.

Using this book

This book is organized into the following chapters:

Chapter 1 *Introduction*

Read this for an introduction to the Cortex-A9 processor and descriptions of the major functional blocks.

Chapter 2 *Functional Description*

Read this for a description of the functionality of the Cortex-A9 processor.

Chapter 3 *Programmers Model*

Read this for a description of the Cortex-A9 registers and programming information.

Chapter 4 *System Control*

Read this for a description of the Cortex-A9 system registers and programming information.

Chapter 5 *Jazelle DBX registers*

Read this for a description of the CP14 coprocessor and its non-debug use for Jazelle DBX.

Chapter 6 *Memory Management Unit*

Read this for a description of the Cortex-A9 *Memory Management Unit* (MMU) and the address translation process.

Chapter 7 *Level 1 Memory System*

Read this for a description of the Cortex-A9 level one memory system, including caches, *Translation Lookaside Buffers* (TLB), and store buffer.

Chapter 8 *Level 2 Memory Interface*

Read this for a description of the Cortex-A9 level two memory interface, the AXI interface attributes, and information about `STRT` instructions.

Chapter 9 *Preload Engine*

Read this for a description of the *Preload Engine* (PLE) and its operations.

Chapter 10 *Debug*

Read this for a description of the Cortex-A9 support for debug.

Chapter 11 *Performance Monitoring Unit*

Read this for a description of the Cortex-A9 *Performance Monitoring Unit* (PMU) and associated events.

Appendix A *Signal Descriptions*

Read this for a summary of the Cortex-A9 signals.

Appendix B *Cycle Timings and Interlock Behavior*

Read this for a description of the Cortex-A9 instruction cycle timing.

Appendix C *Revisions*

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

Conventions

Conventions that this book can use are described in:

- *Typographical*
- *Timing diagrams* on page ix
- *Signals* on page ix.

Typographical

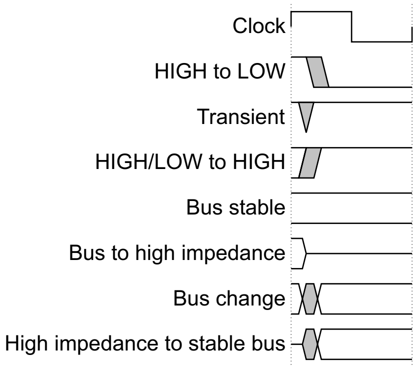
The typographical conventions are:

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 >	Enclose replaceable terms for assembler syntax where they appear in code or code fragments. For example: `MRC p15, 0 <Rd>, <CRn>, <CRm>, <Opcode_2>`

Timing diagrams

The figure named *Key to timing diagram conventions* explains the components used in timing diagrams. Variations, when they occur, have clear labels. You must not assume any timing information that is not explicit in the diagrams.

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



Key to timing diagram conventions

Timing diagrams sometimes show single-bit signals as HIGH and LOW at the same time and they look similar to the bus change shown in *Key to timing diagram conventions*. If a timing diagram shows a single-bit signal in this way then its value does not affect the accompanying description.

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

Additional reading

This section lists publications by ARM and by third parties.

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

See the glossary, `http://infocenter.arm.com/help/topic/com.arm.doc.aeg0014-/index.html`, for a list of terms and acronyms specific to ARM.

ARM publications

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

- *ARM Architecture Reference Manual, ARMv7-A and ARMv7-R edition* (ARM DDI 0406)
- *Cortex-A9 MPCore Technical Reference Manual* (ARM DDI 0407)
- *Cortex-A9 Floating-Point Unit (FPU) Technical Reference Manual* (ARM DDI 0408)

- *Cortex-A9 NEON® Media Processing Engine Technical Reference Manual* (ARM DDI 0409)
- *Cortex-A9 Configuration and Sign-Off Guide* (ARM DII 00146)
- *Cortex-A9 MBIST Controller Technical Reference Manual* (ARM DDI 0414)
- *CoreSight™ PTM-A9 Technical Reference Manual* (ARM DDI 0401)
- *CoreSight PTM-A9 Integration Manual* (ARM DII 0162)
- *CoreSight Program Flow Trace Architecture Specification*,v1.0 (ARM IHI 0035)
- *CoreLink Level 2 Cache Controller L2C-310 Technical Reference Manual* (ARM DDI 0246)
- *AMBA AXI Protocol Specification* (ARM IHI 0022)
- *ARM Generic Interrupt Controller Architecture Specification* (ARM IHI 0048)
- *PrimeCell® Generic Interrupt Controller (PL390) Technical Reference Manual* (ARM DDI 0416)
- *RealView® ICE User Guide* (ARM DUI 0155)
- *CoreSight Architecture Specification* (ARM IHI 0029)
- *CoreSight Technology System Design Guide* (ARM DGI 0012)
- *ARM Debug Interface v5 Architecture Specification* (ARM IHI 0031)

Other publications

- *ANSI/IEEE Std 754-1985, IEEE Standard for Binary Floating-Point Arithmetic*
- *IEEE Std 1500-2005, IEEE Standard Testability Method for Embedded Core-based Integrated Circuits.*

Feedback

ARM welcomes feedback on this product and its documentation.

Feedback on this product

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

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

Feedback on content

If you have comments on content then send an e-mail to `errata@arm.com`. Give:

- the title
- the number, ARM DDI 0388G
- the page numbers to which your comments apply
- a concise explanation of your comments.

ARM also welcomes general suggestions for additions and improvements.

Chapter 1
Introduction

This chapter introduces the Cortex-A9 processor and its features. It contains the following sections:

1.1 About the Cortex-A9 processor

The Cortex-A9 processor is a high-performance, low-power, ARM macrocell with an L1 cache subsystem that provides full virtual memory capabilities. The Cortex-A9 processor implements the ARMv7-A architecture and runs 32-bit ARM instructions, 16-bit and 32-bit Thumb instructions, and 8-bit Java bytecodes in Jazelle state.

Figure 1-1 shows a Cortex-A9 uniprocessor in a design with a PL390 Interrupt Controller and an L2C-310 L2 Cache Controller,

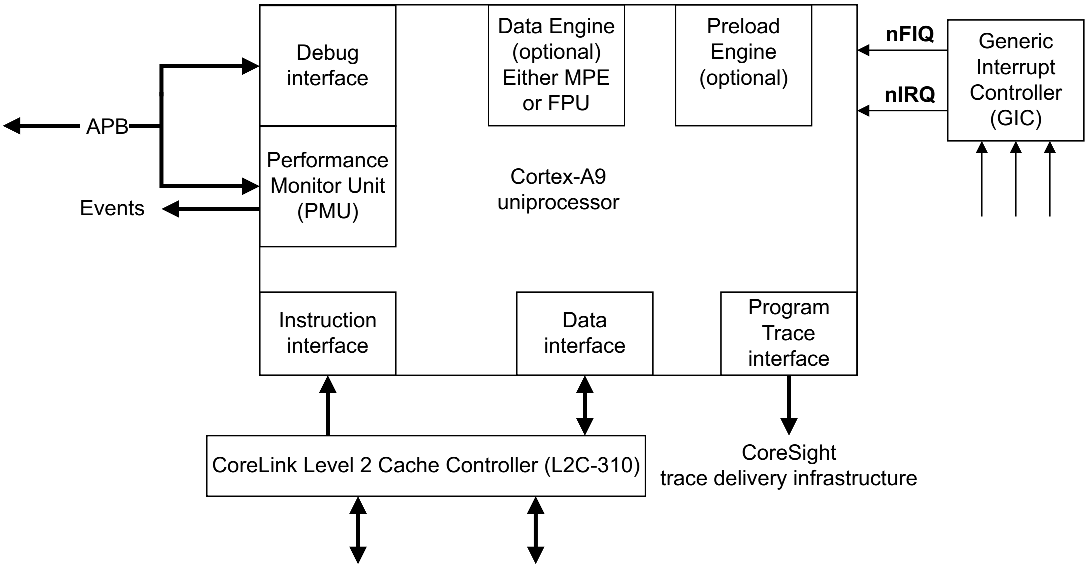


Figure 1-1 Cortex-A9 uniprocessor system

1.1.1 Data engine

The design can include a data engine. The following sections describe the data engine options:

- *Media Processing Engine*
- *Floating-Point Unit*.

Media Processing Engine

The optional *NEON Media Processing Engine* (MPE) is the ARM *Advanced Single Instruction Multiple Data* (SIMD) media processing engine extension to the ARMv7-A architecture. It provides support for integer and floating-point vector operations. NEON MPE can accelerate the performance of multimedia applications such as 3-D graphics and image processing.

When implemented, the NEON MPE option extends the processor functionality to provide support for the ARMv7 Advanced SIMD and VFPv3 D-32 instruction sets.

See the *Cortex-A9 NEON Media Processing Engine Technical Reference Manual*.

Floating-Point Unit

When the design does not include the optional MPE, you can include the optional ARMv7 VFPv3-D16 FPU, without the Advanced SIMD extensions. It provides trapless execution and is optimized for scalar operation. The Cortex-A9 FPU hardware does not support the deprecated VFP short vector feature. Attempts to execute VFP data-processing instructions when the

FPSCR.LEN field is non-zero result in the FPSCR.DEX bit being set and a synchronous Undefined Instruction exception being taken. You can use software to emulate the short vector feature, if required.

See the *Cortex-A9 Floating-Point Unit Technical Reference Manual*.

1.1.2 System design components

This section describes the PrimeCell components in:

- *PrimeCell Generic Interrupt Controller*
- *CoreLink Level 2 Cache Controller (L2C-310)*.

PrimeCell Generic Interrupt Controller

A generic interrupt controller such as the *PrimeCell Generic Interrupt Controller (PL390)* can be attached to the Cortex-A9 uniprocessor. The Cortex-A9 MPCore contains an integrated interrupt controller that shares the same programmers model as the PL390 although there are implementation-specific differences.

See the *Cortex-A9 MPCore Technical Reference Manual* for a description of the Cortex-A9 MPCore Interrupt Controller.

CoreLink Level 2 Cache Controller (L2C-310)

The addition of an on-chip secondary cache, also referred to as a Level 2 or L2 cache, is a recognized method of improving the performance of ARM-based systems when significant memory traffic is generated by the processor. The CoreLink Level 2 Cache Controller reduces the number of external memory accesses and has been optimized for use with Cortex-A9 processors and Cortex-A9 MPCore processors.

1.2 Cortex-A9 variants

Cortex-A9 processors can be used in both a uniprocessor configuration and multiprocessor configurations.

In the multiprocessor configuration, up to four Cortex-A9 processors are available in a cache-coherent cluster, under the control of a *Snoop Control Unit* (SCU), that maintains L1 data cache coherency.

The Cortex-A9 MPCore multiprocessor has:

- up to four Cortex-A9 processors
- an SCU responsible for:
  - maintaining coherency among L1 data caches
  - *Accelerator Coherency Port* (ACP) coherency operations
  - routing transactions on Cortex-A9 MPCore AXI master interfaces
  - Cortex-A9 uniprocessor accesses to private memory regions.
- an *Interrupt Controller* (IC) with support for legacy ARM interrupts
- a private timer and a private watchdog per processor
- a global timer
- AXI high-speed *Advanced Microprocessor Bus Architecture* version 3 (AMBA 3) L2 interfaces.
- an *Accelerator Coherency Port* (ACP), that is, an optional AXI 64-bit slave port that can be connected to a DMA engine or a noncached peripheral.

See the *Cortex-A9 MPCore Technical Reference Manual* for more information.

The following system registers have Cortex-A9 MPCore uses:

- *Multiprocessor Affinity Register* on page 4-19
- *Auxiliary Control Register* on page 4-27
- *Configuration Base Address Register* on page 4-42.

Some PMU event signals have Cortex-A9 MPCore uses. See *Performance monitoring signals* on page A-14.

1.3 Compliance

The Cortex-A9 processor complies with, or implements, the specifications described in:

- *ARM architecture*
- *Advanced Microcontroller Bus Architecture*
- *Program Flow Trace architecture*
- *Debug architecture*
- *Generic Interrupt Controller architecture*

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

1.3.1 ARM architecture

The Cortex-A9 processor implements the ARMv7-A architecture profile that includes the following architecture extensions:

- Advanced *Single Instruction Multiple Data* (SIMD) architecture extension for integer and floating-point vector operations
- *Vector Floating-Point version 3* (VFPv3) architecture extension for floating-point computation that is fully compliant with the IEEE 754 standard
- Security Extensions for enhanced security
- Multiprocessing Extensions for multiprocessing functionality.

See the *ARM Architecture Reference Manual, ARMv7-A and ARMv7-R edition*.

1.3.2 Advanced Microcontroller Bus Architecture

The Cortex-A9 processor complies with the AMBA 3 protocol. See the *AMBA AXI Protocol Specification*.

1.3.3 Program Flow Trace architecture

The Cortex-A9 processor implements the *Program Trace Macrocell* (PTM) based on the *Program Flow Trace* (PFT) v1.0 architecture. See the *CoreSight Program Flow Trace Architecture Specification*.

1.3.4 Debug architecture

The Cortex-A9 processor implements the ARMv7 Debug architecture that includes support for Security Extensions and CoreSight. See the *CoreSight Architecture Specification*.

1.3.5 Generic Interrupt Controller architecture

The Cortex-A9 processor implements the ARM *Generic Interrupt Controller* (GIC) v1.0 architecture.

1.4 Features

The Cortex-A9 processor includes the following features:

- superscalar, variable length, out-of-order pipeline with dynamic branch prediction
- full implementation of the ARM architecture v7-A instruction set
- Security Extensions
- Harvard level 1 memory system with *Memory Management Unit* (MMU).
- two 64-bit AXI master interfaces with Master 0 for the data side bus and Master 1 for the instruction side bus
- ARMv7 Debug architecture
- support for trace with the *Program Trace Macrocell* (PTM) interface
- support for advanced power management with up to three power domains
- optional Preload Engine
- optional Jazelle hardware acceleration
- optional data engine with MPE and VFPv3.

1.5 Interfaces

The processor has the following external interfaces:

- AMBA AXI interfaces
- Debug v7 compliant interface, including a debug APBv3 external debug interface
- DFT.

For more information on these interfaces see:

- *AMBA AXI Protocol Specification*
- *CoreSight Architecture Specification*
- *Cortex-A9 MBIST Controller Technical Reference Manual*

1.6 Configurable options

Table 1-1 shows the configurable options for the Cortex-A9 processor.

Table 1-1 Configurable options for the Cortex-A9 processor

Feature	Range of options	Default value
Instruction cache size	16KB, 32KB, or 64KB	32KB
Data cache size	16KB, 32KB, or 64KB	32KB
TLB entries	64 entries or 128 entries	128 entries
Jazelle Architecture Extension	Full or trivial	Full
Media Processing Engine with NEON technology	Included or not[a]	Not included
FPU	Included or not[a]	
PTM interface	Included or not	
Wrappers for power off and dormant modes	Included or not	
Support for parity error detection	-	Inclusion of this feature is a configuration and design decision.
Preload Engine	Included or not	
Preload Engine FIFO size[b]	16, 8, or 4 entries	16 entries
ARM_BIST	Included or not	Included
USE DESIGNWARE	Use or not	Use

[a] The MPE and FPU RTL options are mutually exclusive. If you choose the MPE option, the MPE is included along with its VFPv3-D32 FPU, and the FPU RTL option is not available in this case. When the MPE RTL option is not implemented, you can implement the VFPv3-D16 FPU by choosing the FPU RTL option.

[b] Only when the design includes the Preload Engine.

The MBIST solution must be configured to match the chosen Cortex-A9 cache sizes. In addition, the form of the MBIST solution for the RAM blocks in the Cortex-A9 design must be determined when the processor is implemented.

See the *Cortex-A9 MBIST Controller Technical Reference Manual* for more information.

1.7 Test features

The Cortex-A9 processor provides test signals that enable the use of both ATPG and MBIST to test the Cortex-A9 processor and its memory arrays. See Appendix A *Signal Descriptions* and the *Cortex-A9 MBIST Controller Technical Reference Manual*.

1.8 Product documentation and design flow

This section describes the Cortex-A9 processor books, and how they relate to the design flow in:

- *Documentation*
- *Design flow* on page 1-11.

See *Additional reading* on page ix for more information about the books described in this section. For information about the relevant architectural standards and protocols, see *Compliance* on page 1-5.

1.8.1 Documentation

The Cortex-A9 documentation is as follows:

Technical Reference Manual

The *Technical Reference Manual* (TRM) describes the functionality and the effects of functional options on the behavior of the Cortex-A9 family of processors. 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. The following TRMs are available with the Cortex-A9 deliverables:

- the *Cortex-A9 TRM* describes the uniprocessor variant.
- the *Cortex-A9 MPCore TRM* describes the multiprocessor variant of the Cortex-A9 processor.
- the *Cortex-A9 Floating-Point Unit (FPU) TRM* describes the implementation-specific FPU parts of the data engine.
- the *Cortex-A9 NEON Media Processing Engine TRM* describes the Advanced SIMD Cortex-A9 implementation-specific parts of the data engine.

If you are programming the Cortex-A9 processor then contact:

- the implementer to determine:
  - the build configuration of the implementation
  - what integration, if any, was performed before implementing the Cortex-A9 processor.
- the integrator to determine the pin configuration of the device 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) source files with the build configuration options
- how to integrate RAM arrays
- how to run test vectors
- 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 documentation from your EDA tools vendor complements the CSG.

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

1.8.2 Design flow

The Cortex-A9 processor is delivered as synthesizable RTL. Before the processor can be used in a product, it must go through the following process:

Implementation

The implementer configures and synthesizes the RTL to produce a hard macrocell. If appropriate, this includes integrating the 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 Cortex-A9 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-A9 processor: The operation of the final device depends on:

  Build configuration

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

  Configuration inputs

  The integrator configures some features of the Cortex-A9 processor by tying inputs to specific values. These configurations affect the start-up behavior before any software configuration is made. They can also limit the options available to the software.

  Software configuration

  The programmer configures the Cortex-A9 processor by programming particular values into registers. This affects the behavior of the Cortex-A9 processor.

Note

This manual refers to implementation-defined features that are applicable to build configuration options. Reference to a feature that is *included* mean that the appropriate build and pin configuration options have been selected. References to an *enabled* feature means that the feature has also been configured by software.

1.9 Product revisions

This section summarizes the differences in functionality between the different releases of this processor:

- *Differences in functionality between r0p0 and r0p1*
- *Differences in functionality between r0p1 and r1p0*
- *Differences in functionality between r1p0 and r2p0* on page 1-13.
- *Differences in functionality between r2p0 and r2p1* on page 1-13.
- *Differences in functionality between r2p1 and r2p2* on page 1-13.
- *Differences in functionality between r2p2 and r3p0* on page 1-13.

1.9.1 Differences in functionality between r0p0 and r0p1

There is no change in the described functionality between r0p0 and r0p1.

The only differences between the two revisions are:

- r0p1 includes fixes for all known engineering errata relating to r0p0
- r0p1 includes an upgrade of the micro TLB entries from 8 to 32 entries, on both the Instruction and Data side.

Neither of these changes affect the functionality described in this document.

1.9.2 Differences in functionality between r0p1 and r1p0

The differences between the two revisions are:

- r1p0 includes fixes for all known engineering errata relating to r0p1.
- In r1p0 **CPUCLKOFF** and **DECLKOFF** enable control of Cortex-A9 processors during reset sequences. See *Configuration signals* on page A-5.
  - In a multiprocessor implementation of the design there are as many **CPUCLKOFF** pins as there are Cortex-A9 processors.
  - **DECLKOFF** controls the data engine clock during reset sequences.
- r1p0 includes dynamic high level clock gating of the Cortex-A9 processor. See Dynamic high level clock gating on page 2-8.
  - **MAXCLKLATENCY[2:0]** bus added. See *Configuration signals* on page A-5
  - Addition of CP15 power control register. See *Power Control Register* on page 4-41.
- Extension of the Performance Monitoring Event bus. In r1p0, **PMUEVENT** is 52 bits wide:
  - Addition of Cortex-A9 specific events. See Table 2-2 on page 2-5.
  - Event descriptions extended. See Table 2-2 on page 2-5.
- Addition of **PMUSECURE** and **PMUPRIV**. See *Performance monitoring signals* on page A-14.
- Main TLB options for 128 entries or 64 entries. See *TLB Type Register* on page 4-19.
- **DEFLAGS[6:0]** added. See DEFLAGS[6:0] on page 4-37.
- The power management signal **BISTSCLAMP** is removed.
- The scan test signal **SCANTEST** is removed.

- Addition of a second replacement strategy. Selection done by SCTLR.RR bit. See *System Control Register* on page 4-24.
- Addition of PL310 cache controller optimization description. See *Optimized accesses to the L2 memory interface* on page 8-7.
- Change to the serializing behavior of `DMB`. See *Serializing instructions* on page B-9.
- ID Register values changed to reflect correct revision.

1.9.3 Differences in functionality between r1p0 and r2p0

The differences between the revisions are:

- Addition of optional Preload Engine hardware feature and support.
  - PLE bit added to NSACR. See *Non-secure Access Control Register* on page 4-32.
  - Preload Engine registers added. See *c11 registers* on page 4-10.
  - Preload operations added and MCRR instruction added. See Chapter 9 *Preload Engine*.
  - Addition of Preload Engine events.

    See Performance monitoring on page 2-3, Table 11-5 on page 11-7, and Table A-18 on page A-14.
- Change to voltage domains. See Figure 2-4 on page 2-14.
- NEON Busy Register. See *NEON Busy Register* on page 4-42.
- ID Register values changed to reflect correct revision.

1.9.4 Differences in functionality between r2p0 and r2p1

- None.

1.9.5 Differences in functionality between r2p1 and r2p2

- None. Documentation updates and corrections only. See *Differences between issue D and issue F* on page C-6.

1.9.6 Differences in functionality between r2p2 and r3p0

- Addition of the REVIDR. See *Revision ID register* on page 4-20.

Chapter 2
Functional Description

This chapter describes the functionality of the product. It contains the following sections:

2.1 About the functions

The Cortex-A9 processor is a high-performance, low-power, ARM macrocell with an L1 cache subsystem that provides full virtual memory capabilities.

Figure 2-1 shows a top-level diagram of the Cortex-A9 processor.

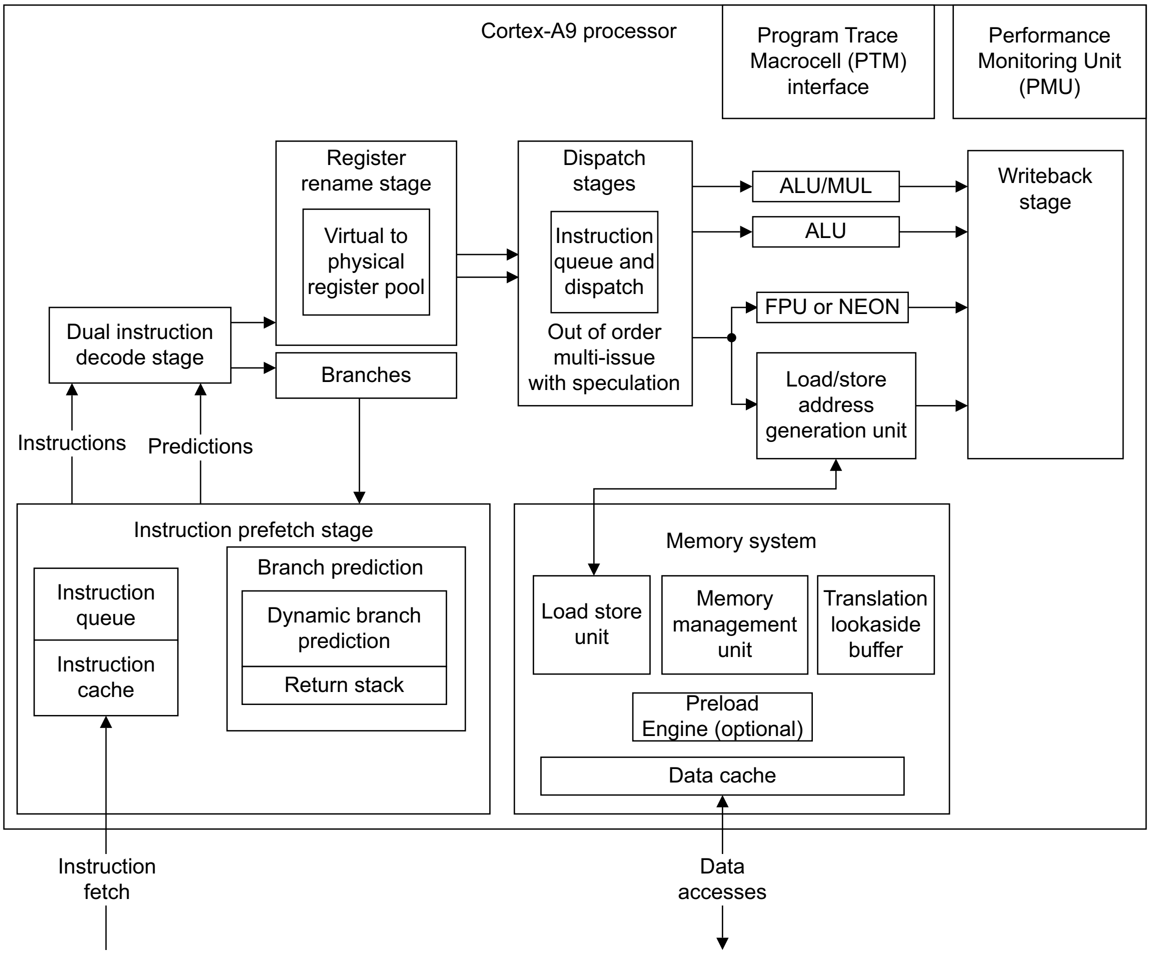


Figure 2-1 Cortex-A9 processor top-level diagram

2.1.1 Instruction queue

In the instruction queue small loop mode provides low power operation while executing small instruction loops. See *Energy efficiency features* on page 2-10.

2.1.2 Dynamic branch prediction

The Prefetch Unit implements 2-level dynamic branch prediction with a *Global History Buffer* (GHB), a *Branch Target Address Cache* (BTAC) and a return stack. See *About the L1 instruction side memory system* on page 7-5.

2.1.3 Register renaming

The register renaming scheme facilitates out-of-order execution in *Write-after-Write* (WAW) and *Write-after-Read* (WAR) situations for the general purpose registers and the flag bits of the Current Program Status Register (CPSR).

The scheme maps the 32 ARM architectural registers to a pool of 56 physical 32-bit registers, and renames the flags (N, Z, C, V, Q, and GE) of the CPSR using a dedicated pool of eight physical 9-bit registers.

2.1.4 PTM interface

The Cortex-A9 processor optionally implements a *Program Trace Macrocell* (PTM) interface, that is compliant with the *Program Flow Trace* (PFT) instruction-only architecture protocol. Waypoints, changes in the program flow or events such as changes in context ID, are output to enable the trace to be correlated with the code image. See *Program Flow Trace and Program Trace Macrocell* on page 2-4.

2.1.5 Performance monitoring

The Cortex-A9 processor provides program counters and event monitors that can be configured to gather statistics on the operation of the processor and the memory system.

You can access performance monitoring counters and their associated control registers from the CP15 coprocessor interface and from the APB Debug interface. See Chapter 11 *Performance Monitoring Unit*.

2.1.6 Virtualization of interrupts

With virtualized interrupts a guest *Operating System* (OS) can use a modified version of the exception behavior model to handle interrupts more efficiently than is possible with a software only solution.

See *Virtualization Control Register* on page 4-34.

The behavior of the Virtualization Control Register depends on whether the processor is in Secure or Non-Secure state.

If the exception occurs when the processor is in Secure state the AMO, IMO and IFO bits in the Virtualization Control Register are ignored. Whether the exception is taken or not depends solely on the setting of the CPSR A, I, and F bits.

If the exception occurs when the processor is in Non-secure state if the SCR EA bit, FIQ bit, or IRQ bit is not set, whether the corresponding exception is taken or not depends solely on the setting of the CPSR A, I, and F bits.

See *Non-secure Access Control Register* on page 4-32.

If the SCR.EAbit, FIQ bit or IRQ bit is set, then the corresponding exception is trapped to Monitor mode. In this case, the corresponding exception is taken or not depending on the CPSR.A bit, I bit, or F bits masked by the AMO, IMO, or IFO bits in the Virtualization Control Register.

2.2 Interfaces

The processor has the following external interfaces:

- *AXI interface*
- *APB external debug interface*
- *Program Flow Trace and Program Trace Macrocell*.

2.2.1 AXI interface

The Cortex-A9 processor implements AMBA 3 AXI interface. See the *AMBA AXI Protocol Specification* for more information.

2.2.2 APB external debug interface

The Cortex-A9 processor implements the ARM Debug interface version 5. See the *CoreSight Architecture Specification* for more information.

2.2.3 Program Flow Trace and Program Trace Macrocell

The Cortex-A9 processor implements the *Program Flow Trace* (PFT) architecture protocol. See the *CoreSight Program Flow Trace Architecture Specification*.

PFT is an instruction-only trace protocol that uses waypoints to correlate the trace to the code image. Waypoints are changes in the program flow or events such as branches or changes in context ID that must be output to enable the trace. See the *CoreSight PTM-A9 Technical Reference Manual* for more information about tracing with waypoints.

Program Trace Macrocell (PTM) is a macrocell that implements the PFT architecture.

Figure 2-2 shows the PTM interface signals.

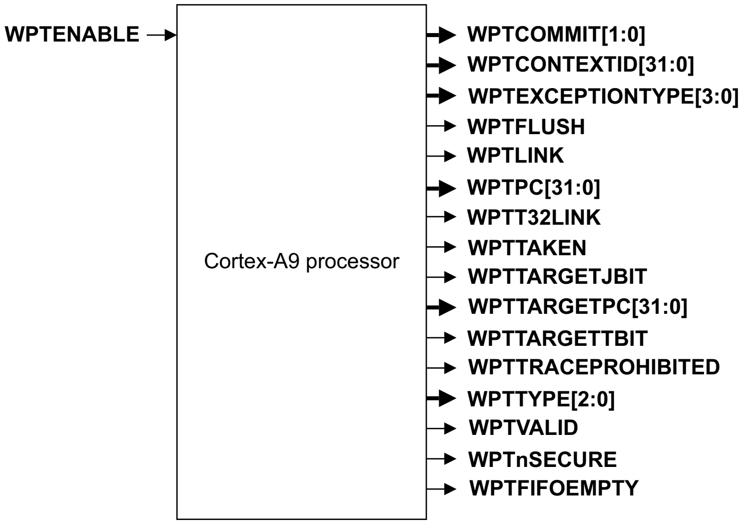


Figure 2-2 PTM interface signals

See Appendix A *Signal Descriptions* and the *CoreSight PTM-A9 Technical Reference Manual* for more information.

Trace must be disabled in some regions. The prohibited regions are described in the *ARM Architecture Reference Manual*. The Cortex-A9 processor must determine prohibited regions for non-invasive debug in regions, including trace, performance monitoring, and PC sampling. No waypoints are generated for instructions that are within a prohibited region.

Note

Only entry to and exit from Jazelle state are traced. A waypoint to enter Jazelle state is followed by a waypoint to exit Jazelle state.

2.3 Clocking and resets

This section describes the clocks and resets of the processor in:

- *Synchronous clocking*
- *Reset*
- *Dynamic high level clock gating* on page 2-8.

2.3.1 Synchronous clocking

The Cortex-A9 uniprocessor has one functional clock input, **CLK**.

The Cortex-A9 uniprocessor does not have any asynchronous interfaces. All the bus interfaces and the interrupt signals must be synchronous with reference to **CLK**.

The AXI bus clock domain can be run at n:1 (AXI: processor ratio to **CLK**) using the **ACLKEN** signal.

Figure 2-3 shows a timing example with **ACKLENM0** used with a 3:1 clock ratio between **CLK** and **ACLK** in a Cortex-A9 uniprocessor.

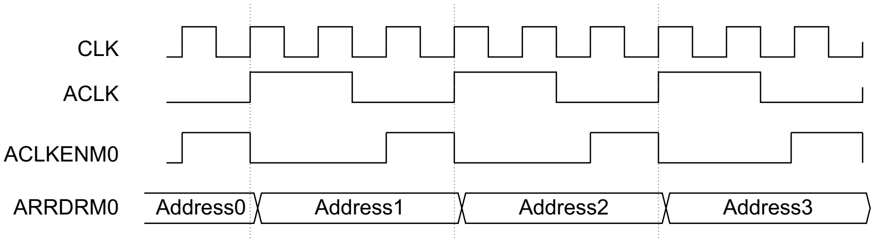


Figure 2-3 ACLKENM0 used with a 3:1 clock ratio

The master port, Master0, changes the AXI outputs only on the **CLK** rising edge when **ACLKENM0** is HIGH.

2.3.2 Reset

The Cortex-A9 processor has the following reset inputs:

nCPURESET	The **nCPURESET** signal is the main Cortex-A9 processor reset. It initializes the Cortex-A9 processor logic and the FPU logic including the FPU register file when the MPE or FPU option is present.
nNEONRESET	The **nNEONRESET** signal is the reset that controls the NEON SIMD independently of the main Cortex-A9 processor reset.
nDBGRESET	The **nDBGRESET** signal is the reset that initializes the debug logic. See Chapter 10 *Debug*.

All of these are active-LOW signals.

Reset modes

The reset signals present in the Cortex-A9 design enable you to reset different parts of the processor independently. Table 2-1 shows the reset signals, and the combinations and possible applications that you can use them in.

Table 2-1 Reset modes

Mode	nCPURESET	nNEONRESET	nDBGRESET
Power-on reset, cold reset	0	0	0
Processor reset, soft or warm reset	0	0	1
SIMD MPE power-on reset	1	0	1
Debug logic reset	1	1	0
No reset, normal run mode	1	1	1

Power-on reset

You must apply power-on or *cold* reset to the Cortex-A9 uniprocessor when power is first applied to the system. In the case of power-on reset, the leading edge, that is the falling edge, of the reset signals do not have to be synchronous to **CLK**, but the rising edge must be.

You must assert the reset signals for at least nine **CLK** cycles to ensure correct reset behavior.

ARM recommends the following reset sequence:

1. Apply **nCPURESET** and **nDBGRESET**, plus **nNEONRESET** if the SIMD MPE is present.
2. Wait for at least nine **CLK** cycles, plus at least one cycle in each other clock domain, or more if the documentation for other components requires it. There is no harm in applying more clock cycles than this, and maximum redundancy can be achieved by applying 15 cycles on every clock domain.
3. Stop the **CLK** clock input to the Cortex-A9 uniprocessor. If there is a data engine present, use **NEONCLKOFF**. See *Configuration signals* on page A-5.
4. Wait for the equivalent of approximately 10 cycles, depending on your implementation. This compensates for clock and reset tree latencies.
5. Release all resets.
6. Wait for the equivalent of another approximately 10 cycles, again to compensate for clock and reset tree latencies.
7. Restart the clock.

Software reset

A processor or warm reset initializes the majority of the Cortex-A9 processor, apart from its debug logic. Breakpoints and watchpoints are retained during a processor reset. Processor reset is typically used for resetting a system that has been operating for some time. Use the same reset sequence described in *Power-on reset* with the only difference that **nDBGRESET** must remain HIGH during the sequence, to ensure that all values in the debug registers are maintained.

Processor reset

A processor or *warm* reset initializes the majority of the Cortex-A9 processor, apart from its debug logic. Breakpoints and watchpoints are retained during a processor reset. Processor reset is typically used for resetting a system that has been operating for some time. Use **nCPURESET** and **nNEONRESET** for a warm reset.

MPE SIMD logic reset

This reset initializes all the SIMD logic of the MPE. It is expected to be applied when the SIMD part of the MPE exits from powerdown state. This reset only applies to configurations where the SIMD MPE logic is implemented in its own dedicated power domain, separated from the rest of the processor logic.

ARM recommends the following reset sequence for an MPE SIMD reset:

1. Apply **nNEONRESET**.
2. Wait for at least nine **CLK** cycles. 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. Assert **NEONCLKOFF** with a value of 1’b1.
4. Wait for the equivalent of approximately 10 cycles, depending on your implementation. This compensates for clock and reset tree latencies.
5. Release **nNEONRESET**.
6. Wait for the equivalent of another approximately 10 cycles, again to compensate for clock and reset tree latencies.
7. Deassert **NEONCLKOFF**. This ensures that all registers in the SIMD MPE part of the processor see the same **CLK** edge on exit from the reset sequence.

Use **nNEONRESET** to control the SIMD part of the MPE logic independently of the Cortex-A9 processor reset. Use this reset to hold the SIMD part of the MPE in a reset state so that the power to the SIMD part of the MPE can be safely switched on or off. See Table 2-2 on page 2-10.

Debug reset

This reset initializes the debug logic in the Cortex-A9 uniprocessor, including breakpoints and watchpoints values.

To perform a debug reset, you must assert the **nDBGRESET** signal LOW during a few **CLK** cycles.

2.3.3 Dynamic high level clock gating

The following sections describe dynamic high level clock gating:

- *Gated blocks* on page 2-9
- *Power Control Register* on page 2-9
- *Dynamic high level clock gating activity* on page 2-9.

Gated blocks

The Cortex-A9 processor or each processor in a CortexA9 MPCore design supports dynamic high level clock gating of:

- the integer core
- the system control block.
- the data engine, if implemented.

Power Control Register

The Power Control Register controls dynamic high level clock gating. This register contains fields that are common to these blocks:

- the enable bit for clock gating
- the max_clk_latency bits.

See *Power Control Register* on page 4-41.

Dynamic high level clock gating activity

When dynamic high level clock gating is enabled the clock of the integer core is cut in the following cases:

- the integer core is empty and there is an instruction miss causing a linefill
- the integer core is empty and there is an instruction TLB miss
- the integer core is full and there is a data miss causing a linefill
- the integer core is full and data stores are stalled because the linefill buffers are busy.

When dynamic clock gating is enabled, the clock of the system control block is cut in the following cases:

- there are no system control coprocessor instructions being executed
- there are no system control coprocessor instructions present in the pipeline
- performance events are not enabled
- debug is not enabled.

When dynamic clock gating is enabled, the clock of the data engine is cut when there is no data engine instruction in the data engine and no data engine instruction in the pipeline.

2.4 Power management

The processor provides mechanisms to control both dynamic and static power dissipation. Static power control is implementation-specific. This section describes:

- *Energy efficiency features*
- *Cortex-A9 processor power control*.
- *Power domains* on page 2-13.
- *Cortex-A9 voltage domains* on page 2-13.

2.4.1 Energy efficiency features

The features of the Cortex-A9 processor that improve energy efficiency include:

- accurate branch and return prediction, reducing the number of incorrect instruction fetch and decode operations
- the use of physically addressed caches, reducing the number of cache flushes and refills, saving energy in the system
- the use of micro TLBs reduces the power consumed in translation and protection lookups for each cycle
- caches that use sequential access information to reduce the number of accesses to the tag RAMs and to unnecessary accesses to data RAMs
- instruction loops that are smaller than 64 bytes often complete without additional instruction cache accesses, so lowering power consumption.

2.4.2 Cortex-A9 processor power control

Place holders for level-shifters and clamps are inserted around the Cortex-A9 processor to ease the implementation of different power domains.

The Cortex-A9 processor can have the following power domains:

- a power domain for Cortex-A9 processor logic
- a power domain for Cortex-A9 processor MPE
- a power domain for Cortex-A9 processor RAMs.

Table 2-2 shows the power modes.

Table 2-2 Cortex-A9 processor power modes

Mode	Cortex-A9 processor RAM arrays	Cortex-A9 processor logic	Cortex-A9 data engine	Description
Full Run Mode	Powered-up	Powered-up	Powered-up	-
		Clocked	Clocked	
Run Mode with MPE disabled	Powered-up	Powered-up	Powered-up	See *Coprocessor Access Control Register* on page 4-29 for information about disabling the MPE
		Clocked	No clock	
Run Mode with MPE powered off	Powered-up	Powered-up	Powered off	The MPE can be implemented in a separate power domain and be powered off separately
		Clocked		

Table 2-2 Cortex-A9 processor power modes (continued)

Mode	Cortex-A9 processor RAM arrays	Cortex-A9 processor logic	Cortex-A9 data engine	Description
Standby	Powered-up	Powered-up	Powered Up	Standby modes, see *Standby modes*
		Only wake-up logic is clocked.	Clock is disabled, or powered off	
Dormant	Retention state/voltage	Powered-off	Powered-off	External wake-up event required to wake up
Shutdown	Powered-off	Powered-off	Powered-off	External wake-up event required to wake up

Entry to Dormant or Shutdown mode must be controlled through an external power controller.

Run mode

Run mode is the normal mode of operation, where all of the functionality of the Cortex-A9 processor is available.

Standby modes

WFI and WFE Standby modes disable most of the clocks in a processor, while keeping its logic powered up. This reduces the power drawn to the static leakage current, leaving a tiny clock power overhead requirement to enable the device to wake up.

Entry into WFI Standby mode is performed by executing the `WFI` instruction.

The transition from the WFI Standby mode to the Run mode is caused by:

- An **IRQ** interrupt, regardless of the value of the CSPR.I bit.
- An **FIQ** interrupt, regardless of the value of the CSPR.F bit.
- An asynchronous abort, regardless of the value of the CPSR.A bit.
- A debug event, if invasive debug is enabled and the debug event is permitted.
- A CP15 maintenance request broadcast by other processors. This applies to the Cortex-A9 MPCore product only.

Entry into WFE Standby mode is performed by executing the `WFE` instruction.

The transition from the WFE Standby mode to the Run mode is caused by:

- An **IRQ** interrupt, unless masked by the CPSR.I bit.
- An **FIQ** interrupt, unless masked by the CPSR.F bit.
- An asynchronous abort, unless masked by the CPSR.A bit.
- A debug event, if invasive debug is enabled and the debug event is permitted.
- The assertion of the **EVENTI** input signal.
- The execution of an `SEV` instruction on any processor in the multiprocessor system. This applies to the Cortex-A9 MPCore product only.
- A CP15 maintenance request broadcast by other processors. This applies to the Cortex-A9 MPCore product only.

The debug request can be generated by an externally generated debug request, using the **EDBGRQ** pin on the Cortex-A9 processor, or from a Debug Halt instruction issued to the Cortex-A9 processor through the APB debug port.

The debug channel remains active throughout a `WFI` instruction.

Dormant mode

Dormant mode enables the Cortex-A9 processor to be powered down, while leaving the caches powered up and maintaining their state.

The RAM blocks that must remain powered up during Dormant mode are:

- all data RAMs associated with the cache
- all tag RAMs associated with the cache
- outer RAMs.

The RAM blocks that are to remain powered up must be implemented on a separate power domain.

Before entering Dormant mode, the state of the Cortex-A9 processor, excluding the contents of the RAMs that remain powered up in dormant mode, must be saved to external memory. These state saving operations must ensure that the following occur:

- All ARM registers, including CPSR and SPSR registers are saved.
- All system registers are saved.
- All debug-related state must be saved.
- A Data Synchronization Barrier instruction is executed to ensure that all state saving has completed.
- The Cortex-A9 processor then communicates with the power controller, using the **STANDBYWFI**, to indicate that it is ready to enter dormant mode by performing a `WFI` instruction. See *Communication to the power management controller* on page 2-13 for more information.
- Before removing the power, the reset signals to the Cortex-A9 processor must be asserted by the external power control mechanism.

The external power controller triggers the transition from Dormant state to Run state. The external power controller must assert reset to the Cortex-A9 processor until the power is restored. After power is restored, the Cortex-A9 processor leaves reset and can determine that the saved state must be restored.

Shutdown mode

Shutdown mode powers down the entire device, and all state, including cache, must be saved externally by software. This state saving is performed with interrupts disabled, and finishes with a Data Synchronization Barrier operation. The Cortex-A9 processor then communicates with a power controller that the device is ready to be powered down in the same manner as when entering Dormant Mode. The processor is returned to the run state by asserting reset.

Note

You must power up the processor before performing a reset.

Communication to the power management controller

Communication between the Cortex-A9 processor and the external power management controller can be performed using the Standby signals, Cortex-A9 input clamp signals, and **DBGNOPWRDWN**.

Standby signals

These signals control the external power management controller.

The **STANDBYWFI** signal indicates that the Cortex-A9 processor is ready to enter Power Down mode. See *WFE and WFI standby signals* on page A-6.

Cortex-A9 input signals

The external power management controller uses **NEONCLAMP** and **CPURAMCLAMP** to isolate Cortex-A9 power domains from one another before they are turned off. These signals are only meaningful if the Cortex-A9 processor implements power domain clamps. See *Power management signals* on page A-7.

DBGNOPWRDWN

DBGNOPWRDWN is connected to the system power controller and is interpreted as a request to operate in emulate mode. In this mode, the Cortex-A9 processor and PTM are not actually powered down when requested by software or hardware handshakes. See *Miscellaneous debug interface signals* on page A-23.

2.4.3 Power domains

The Cortex-A9 uniprocessor contains optional placeholders between the Cortex-A9 logic and RAM arrays, or between the Cortex-A9 logic and the NEON SIMD logic, when NEON is present, so that these parts can be implemented in different voltage domains.

2.4.4 Cortex-A9 voltage domains

The Cortex-A9 processor can have the following power domains:

- Cortex-A9 processor logic cells
- Cortex-A9 processor data engines
- Cortex-A9 processor RAMs.

Figure 2-4 on page 2-14 shows the power domains.

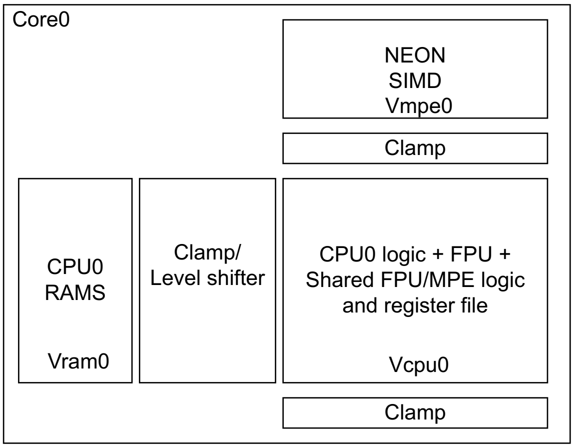


Figure 2-4 Power domains for the Cortex-A9 processor

The FPU is part of the processor power domain. The FPU clock is based on the processor clock. There is static and dynamic high-level clock-gating. NEON SIMD data paths and logic are in a separate power domain, with dedicated clock and reset signals. There is static and dynamic high-level clock-gating.

When NEON is present, you can run FPU (non-SIMD) code without powering the SIMD part or clocking the SIMD part.

2.5 Constraints and limitations of use

This section describes memory consistency.

Memory coherency in a Cortex-A9 processor is maintained following a weakly ordered memory consistency model.

Note

When the Shareable attribute is applied to a memory region that is not Write-Back, Normal memory, data held in this region is treated as Non-cacheable.

Chapter 3

Programmers Model

This chapter describes the processor registers and provides information for programming the processor. It contains the following sections:

3.1 About the programmers model

The Cortex-A9 processor implements the ARMv7-A architecture.

See the *ARM Architecture Reference Manual* for information about the ARMv7-A architecture.

3.2 ThumbEE architecture

The *Thumb Execution Environment* (ThumbEE) extension is a variant of the Thumb instruction set that is designed as a target for dynamically generated code. See the *ARM Architecture Reference Manual* for more information.

3.3 The Jazelle Extension

The Cortex-A9 processor provides hardware support for the Jazelle Extension. The processor accelerates the execution of most bytecodes. Some bytecodes are executed by software routines.

See the *ARM Architecture Reference Manual* for more information.

See Chapter 5 *Jazelle DBX registers*.

3.4 Advanced SIMD architecture

The Advanced SIMD architecture extension is a media and signal processing architecture that adds instructions targeted primarily at audio, video, 3-D graphics, image, and speech processing.

Note

The Advanced SIMD architecture extension, its associated implementations, and supporting software, are commonly referred to as NEON MPE.

NEON MPE includes both Advanced SIMD instructions and the ARM VFPv3 instructions. All Advanced SIMD instructions and VFP instructions are available in both ARM and Thumb states.

See the *ARM Architecture Reference Manual* for more information.

See the *Cortex-A9 NEON Media Processing Engine Technical Reference Manual* for implementation-specific information.

3.5 Security Extensions architecture

Security Extensions enable the construction of a secure software environment. This section describes the following:

- *System boot sequence*.

See the *ARM Architecture Reference Manual* for more information.

3.5.1 System boot sequence

Caution

The Security Extensions enable the construction of an isolated software environment for more secure execution, depending on a suitable system design around the processor. The technology does not protect the processor from hardware attacks, and you must ensure that the hardware containing the reset handling code is appropriately secure.

The processor always boots in the Privileged Supervisor mode in the Secure state, with the NS bit set to 0. This means that code that does not attempt to use the Security Extensions always runs in the Secure state. If the software uses both Secure and Non-secure states, the less trusted software, such as a complex operating system, executes in Non-secure state, and the more trusted software executes in the Secure state.

The following sequence is expected to be typical use of the Security Extensions:

1. Exit from reset in Secure state.
2. Configure the security state of memory and peripherals. Some memory and peripherals are accessible only to the software running in Secure state.
3. Initialize the secure operating system. The required operations depend on the operating system, and typically include initialization of caches, MMU, exception vectors, and stacks.
4. Initialize Secure Monitor software to handle exceptions that switch execution between the Secure and Non-Secure operating systems.
5. Optionally lock aspects of the secure state environment against additional configuration.
6. Pass control through the Secure Monitor software to the Non-Secure OS with an `SMC` instruction to enable the Non-secure operating system to initialize. The required operations depend on the operating system, and typically include initialization of caches, MMU, exception vectors, and stacks.

The overall security of the secure software depends on the system design, and on the secure software itself.

3.6 Multiprocessing Extensions

The Multiprocessing Extensions are a set of features that enhance multiprocessing functionality. See the *ARM Architecture Reference Manual* for more information.

3.7 Modes of operation and execution

This section describes the instruction set states and modes of the Cortex-A9 processor in:

- *Operating states*.

3.7.1 Operating states

The processor has the following instruction set states controlled by the T bit and J bit in the CPSR.

ARM state	The processor executes 32-bit, word-aligned ARM instructions.
Thumb state	The processor executes 16-bit and 32-bit, halfword-aligned Thumb instructions.
Jazelle state	The processor executes variable length, byte-aligned Jazelle instructions.
ThumbEE state	The processor executes a variant of the Thumb instruction set designed as a target for dynamically generated code. This is code compiled on the device either shortly before or during execution from a portable bytecode or other intermediate or native representation.

The J bit and the T bit determine the instruction set used by the processor. Table 3-1 shows the encoding of these bits.

Table 3-1 CPSR J and T bit encoding

J	T	Instruction set state
0	0	ARM
0	1	Thumb
1	0	Jazelle
1	1	ThumbEE

Note

Transition between ARM and Thumb states does not affect the processor mode or the register contents. See the *ARM Architecture Reference Manual* for information on entering and exiting ThumbEE state.

3.8 Memory model

The Cortex-A9 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 processor can store words in memory in either big-endian format or little-endian format.

Instructions are always treated as little-endian.

Note

ARMv7 does not support the BE-32 memory model.

3.9 Addresses in the Cortex-A9 processor

In the Cortex-A9 processor, the VA and MVA are identical.

When the Cortex-A9 processor is executing in Non-secure state, the processor performs translation table lookups using the Non-secure versions of the Translation Table Base Registers. In this situation, any VA can only translate into a Non-secure PA.When it is in Secure state, the Cortex-A9 processor performs translation table lookups using the Secure versions of the Translation Table Base Registers. In this situation, the security state of any VA is determined by the NS bit of the translation table descriptors for that address.

Table 3-2 shows the address types in the processor system.

Table 3-2 Address types in the processor system

Processor	Caches	Translation Lookaside Buffers	AXI bus
Data VA	Data cache is *Physically Indexed Physically Tagged* (PIPT)	Translates Virtual Address to Physical Address	Physical Address
Instruction VA	Instruction cache is *Virtually Indexed Physically Tagged* (VIPT)		

This is an example of the address manipulation that occurs when the Cortex-A9 processor requests an instruction.

1. The Cortex-A9 processor issues the VA of the instruction as Secure or Non-secure VA according to the state the processor is in.
2. The instruction cache is indexed by the lower bits of the VA. The TLB performs the translation in parallel with the cache lookup. The translation uses Secure descriptors if the processor is in the Secure state. Otherwise it uses the Non-secure descriptors.
3. If the protection check carried out by the TLB on the VA does not abort and the PA tag is in the instruction cache, the instruction data is returned to the processor.
4. If there is a cache miss, the PA is passed to the AXI bus interface to perform an external access. The external access is always Non-secure when the processor is in the Non-secure state. In the Secure state, the external access is Secure or Non-secure according to the NS attribute value in the selected descriptor. In Secure state, both L1 and L2 table walks accesses are marked as Secure, even if the first level descriptor is marked as NS.

   Note

   Secure L2 lookups are secure even if the L1 entry is marked Non-secure.

Chapter 4

System Control

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

- *About system control* on page 4-2
- *Register summary* on page 4-3
- *Register descriptions* on page 4-18.

4.1 About system control

The system control coprocessor, CP15, controls and provides status information for the functions implemented in the processor. The main functions of the system control coprocessor are:

- overall system control and configuration
- MMU configuration and management
- cache configuration and management
- system performance monitoring.

4.1.1 Deprecated registers

In ARMv7-A the following have instruction set equivalents:

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

The use of the registers is optional and deprecated.

In addition, the Fast Context Switch Extensions are deprecated in ARM v7 architecture, and are not implemented in the Cortex-A9 processor.

4.2 Register summary

This section gives a summary of the CP15 system control registers. For more information on using the CP15 system control registers, see the *ARM Architecture Reference Manual.*

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

This section describes the CP15 system control registers grouped by CRn order, and accessed by the `MCR` and `MRC` instructions in the order of CRn, Op1, CRm, Op2:

- *c0 registers* on page 4-5
- *c1 registers* on page 4-6
- *c2 registers* on page 4-6
- *c3 registers* on page 4-6
- *c4 registers* on page 4-6
- *c5 registers* on page 4-7
- *c6 registers* on page 4-7
- *c7 registers* on page 4-7
- *c8 registers* on page 4-8
- *c9 registers* on page 4-9
- *c10 registers* on page 4-9
- *c11 registers* on page 4-10
- *c12 registers* on page 4-10
- *c13 registers* on page 4-10
- *c14 registers* on page 4-11
- *c15 registers* on page 4-11.

All system control coprocessor registers are 32 bits wide, except for the Program New Channel operation described in *PLE Program New Channel operation* on page 9-5. Reserved registers are RAZ/WI.

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

- *Identification Registers* on page 4-11
- *Virtual memory control registers* on page 4-13
- *Fault handling registers* on page 4-13
- *Other system control registers* on page 4-13
- *Cache maintenance operations* on page 4-14
- *Address translation operations* on page 4-14
- *Miscellaneous operations* on page 4-14
- *Performance monitor registers* on page 4-15
- *Security Extensions registers* on page 4-15
- *Preload Engine registers* on page 4-16
- *TLB maintenance* on page 4-16
- *Implementation defined registers* on page 4-17.

Table 4-1 describes the column headings that the CP15 register summary tables use throughout this section.

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

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

4.2.1 c0 registers

Table 4-2 shows the CP15 system control registers you can access when CRn is c0.

Table 4-2 c0 register summary

Op1	CRm	Op2	Name	Type	Reset	Description
0	c0	0	MIDR	RO	Product revision dependant	*Main ID Register* on page 4-18
		1	CTR	RO	`0x83338003`	Cache Type Register
		2	TCMTR	RO	`0x00000000`	TCM Type Register
		3	TLBTR[a]	RO	-	*TLB Type Register* on page 4-19
		5	MPIDR	RO	-	*Multiprocessor Affinity Register* on page 4-19
		6	REVIDR	RO	-	*Revision ID register* on page 4-20
	c1	0	ID_PFR0	RO	`0x00001231`	Processor Feature Register 0
		1	ID_PFR1	RO	`0x00000011`	Processor Feature Register 1
		2	ID_DFR0	RO	`0x00010444`	Debug Feature Register 0
		3	ID_AFR0	RO	`0x00000000`	Auxiliary Feature Register 0
		4	ID_MMFR0	RO	`0x00100103`	Memory Model Feature Register 0
		5	ID_MMFR1	RO	`0x20000000`	Memory Model Feature Register 1
		6	ID_MMFR2	RO	`0x01230000`	Memory Model Feature Register 2
		7	ID_MMFR3	RO	`0x00102111`	Memory Model Feature Register 3
	c2	0	ID_ISAR0	RO	`0x00101111`	Instruction Set Attributes Register 0
		1	ID_ISAR1	RO	`0x13112111`	Instruction Set Attributes Register 1
		2	ID_ISAR2	RO	`0x21232041`	Instruction Set Attributes Register 2
		3	ID_ISAR3	RO	`0x11112131`	Instruction Set Attributes Register 3
		4	ID_ISAR4	RO	`0x00011142`	Instruction Set Attributes Register 4
1	c0	0	CCSIDR	RO	-	*Cache Size Identification Register* on page 4-21
		1	CLIDR	RO	`0x09000003`	*Cache Level ID Register* on page 4-22
		7	AIDR	RO	`0x00000000`	*Auxiliary ID Register* on page 4-23
2	c0	0	CSSELR	RW	-	*Cache Size Selection Register* on page 4-24

[a] Depends on TLBSIZE. See *TLB Type Register* on page 4-19.

4.2.2 c1 registers

Table 4-3 shows the CP15 system control registers you can access when CRn is c1.

Table 4-3 c1 register summary

Op1	CRm	Op2	Name	Type	Reset	Description
0	c0	0	SCTLR	RW	-[a]	*System Control Register* on page 4-24
		1	ACTLR[b]	RW	`0x00000000`	*Auxiliary Control Register* on page 4-27
		2	CPACR	RW	-[c]	*Coprocessor Access Control Register* on page 4-29
	c1	0	SCR[d]	RW	`0x00000000`	Secure Configuration Register
		1	SDER[c]	RW	`0x00000000`	*Secure Debug Enable Register* on page 4-31
		2	NSACR	RW[e]	-[f]	*Non-secure Access Control Register* on page 4-32
		3	VCR[c]	RW	`0x00000000`	*Virtualization Control Register* on page 4-34

a. Depends on input signals. See *System Control Register* on page 4-24.

b. RO in Non-secure state if NSACR[18]=0 and RW if NSACR[18]=1.

c. `0x00000000` if NEON present and `0xC0000000` if NEON not present or powered down.

d. No access in Non-secure state.

e. RW in Secure state and RO in the Non-secure state.

f. `0x00000000` if NEON present and `0x0000C000` if NEON not present.

4.2.3 c2 registers

Table 4-4 shows the CP15 system control registers you can access when CRn is c2.

Table 4-4 c2 register summary

Op1	CRm	Op2	Name	Type	Reset	Description
0	c0	0	TTBR0	RW	-	
		1	TTBR1	RW	-	Translation Table Base Register 1
		2	TTBCR	RW	`0x00000000`[a]	Translation Table Base Control Register

a. In Secure state only. You must program the Non-secure version with the required value.

4.2.4 c3 registers

Table 4-5 shows the CP15 system control registers you can access when CRn is c3.

Table 4-5 c3 register summary

Op1	CRm	Op2	Name	Type	Reset	Description
0	c0	0	DACR	RW	-	Domain Access Control Register

4.2.5 c4 registers

No CP15 system control registers are accessed with CRn set to c4.

4.2.6 c5 registers

Table 4-6 shows the CP15 system control registers you can access when CRn is c5.

Table 4-6 c5 register summary

Op1	CRm	Op2	Name	Type	Reset	Description
0	c0	0	DFSR	RW	-	Data Fault Status Register
		1	IFSR	RW	-	Instruction Fault Status Register
	c1	0	ADFSR	-	-	Auxiliary Data Fault Status Register
		1	AIFSR	-	-	Auxiliary Instruction Fault Status Register

4.2.7 c6 registers

Table 4-7 shows the CP15 system control registers you can access when CRn is c6.

Table 4-7 c6 register summary

Op1	CRm	Op2	Name	Type	Reset	Description
0	c0	0	DFAR	RW	-	Data Fault Address Register
		2	IFAR	RW	-	Instruction Fault Address Register

4.2.8 c7 registers

Table 4-8 shows the CP15 system control registers you can access when CRn is c7.

Table 4-8 c7 register summary

Op1	CRm	Op2	Name	Type	Reset	Description
0	c0	0-3	Reserved	WO	-	-
		4	NOP[a]	WO	-	-
	c1	0	ICIALLUIS	WO	-	Cache operations registers
		6	BPIALLIS	WO	-	
		7	Reserved	WO	-	
	c4	0	PAR	RW	-	-
	c5	0	ICIALLU	WO	-	Cache operations registers
		1	ICIMVAU	WO	-	
		2-3	Reserved	WO	-	
		4	ISB	WO	User	*Deprecated registers* on page 4-2
		6	BPIALL	WO	-	Cache operations registers
	c6	1	DCIMVAC	WO	-	
		2	DCISW	WO	-	

Table 4-8 c7 register summary (continued)

Op1	CRm	Op2	Name	Type	Reset	Description
0	c8	0-7	V2PCWPR	WO	-	VA to PA operations
	c10	1	DCCVAC	WO	-	Cache operations registers
		2	DCCSW	WO	-	
		4	DSB	WO	User	*Deprecated registers* on page 4-2
		5	DMB	WO	User	
	c11	1	DCCVAU	WO	-	Cache operations registers
	c14	1	DCCIMVAC	WO	-	
		2	DCCISW	WO	-	

a. This operation is performed by the `WFI` instruction. See *Deprecated registers* on page 4-2.

4.2.9 c8 registers

Table 4-9 shows the CP15 system control registers you can access when CRn is c8.

Table 4-9 c8 register summary

Op1	CRm	Op2	Name	Type	Reset	Description
0	c3	0	TLBIALLIS[a]	WO	-	-
		1	TLBIMVAIS[b]	WO	-	-
		2	TLBIASIDIS[b]	WO	-	-
		3	TLBIMVAAIS[a]	WO	-	-
	c5, c6, or c7	0	TLBIALL[a]	WO	-	-
		1	TLBIMVA[b]	WO	-	-
		2	TLBIASID[b]	WO	-	-
		3	TLBIMVAA[a]	WO	-	-

a. Has no effect on entries that are locked down.

b. Invalidates the locked entry when it matches.

See *Invalidate TLB Entries on ASID Match* on page 4-45.

4.2.10 c9 registers

Table 4-10 shows the CP15 system control registers you can access when CRn is c9.

Table 4-10 c9 register summary

Op1	CRm	Op2	Name	Type	Reset	Description
0	c12	0	PMCR	RW	`0x41093000`	Performance Monitor Control Register
		1	PMCNTENSET	RW	`0x00000000`	Count Enable Set Register
		2	PMCNTENCLR	RW	`0x00000000`	Count Enable Clear Register
		3	PMOVSR	RW	-	Overflow Flag Status Register
		4	PMSWINC	WO	-	Software Increment Register
		5	PMSELR	RW	`0x00000000`	Event Counter Selection Register
	c13	0	PMCCNTR	RW	-	Cycle Count Register
		1	PMXEVTYPER	RW	-	Event Type Selection Register
		2	PMXEVCNTR	RW	-	Event Count Registers
	c14	0	PMUSERENR	RW[a]	`0x00000000`	User Enable Register
		1	PMINTENSET	RW	`0x00000000`	Interrupt Enable Set Register
		2	PMINTENCLR	RW	`0x00000000`	Interrupt Enable Clear Register

[a] RO in User mode.

See Chapter 11 *Performance Monitoring Unit*.

4.2.11 c10 registers

Table 4-11 shows the CP15 system control registers you can access when CRn is c10.

Table 4-11 c10 register summary

Op1	CRm	Op2	Name	Type	Reset	Description
0	c0	0	TLB Lockdown Register[a]	RW	`0x00000000`	*TLB Lockdown Register* on page 4-35
	c2	0	PRRR	RW	`0x00098AA4`	Primary Region Remap Register
		1	NRRR	RW	`0x44E048E0`	Normal Memory Remap Register

[a] No access in Non-secure state if NSCAR.TL=0 and RW if NSACR.TL=1.

4.2.12 c11 registers

Table 4-12 shows the CP15 system control registers you can access where CRn is c11.

Table 4-12 c11 register summary

Op1	CRm	Op2	Name	Type	Reset	Description
0	c0	0	PLEIDR	RO[a]	-	*PLE ID Register* on page 4-36
		2	PLEASR	RO[a]	-	*PLE Activity Status Register* on page 4-36
		4	PLEFSR	RO[a]	-	*PLE FIFO Status Register* on page 4-37
	c1	0	PLEUAR	Privileged R/W User RO	-	*Preload Engine User Accessibility Register* on page 4-38
		1	PLEPCR	Privileged R/W User RO	-	*Preload Engine Parameters Control Register* on page 4-39

a. RAZ if the PLE is not present.

4.2.13 c12 registers

Table 4-13 shows the CP15 system control registers you can access when CRn is c12.

Table 4-13 c12 register summary

Op1	CRm	Op2	Name	Type	Reset	Description
0	c0	0	VBAR	RW	`0x00000000`[a]	Vector Base Address Register
		1	MVBAR	RW	-	Monitor Vector Base Address Register
	c1	0	ISR	RO	`0x00000000`	Interrupt Status Register
		1	Virtualization Interrupt Register	RW	`0x00000000`	*Virtualization Interrupt Register* on page 4-40

a. Only the secure version is reset to 0. The Non-secure version must be programmed by software.

4.2.14 c13 registers

Table 4-14 shows the CP15 system control registers you can access when CRn is c13.

Table 4-14 c13 register summary

Op1	CRm	Op2	Name	Type	Reset	Description
0	c0	0	FCSEIDR	RW	`0x00000000`	*Deprecated registers* on page 4-2
		1	CONTEXTIDR	RW	-	Context ID Register
		2	TPIDRURW	RW[a]	-	Software Thread ID registers
		3	TPIDRURO	RO[b]	-	
		4	TPIDRPRW.	RW	-	

a. RW in User mode.

b. RO in User mode.

4.2.15 c14 registers

No CP15 system control registers are accessed with CRn set to c14.

4.2.16 c15 registers

Table 4-15 shows the CP15 system control registers you can access when CRn is c15.

Table 4-15 c15 system control register summary

Op1	CRm	Op2	Name	Type	Reset	Description
0	c0	0	Power Control Register	RW[a]	- [b]	*Power Control Register* on page 4-41
	c1	0	NEON Busy Register	RO	`0x00000000`	*NEON Busy Register* on page 4-42
4	c0	0	Configuration Base Address	RO[c]	- [d]	*Configuration Base Address Register* on page 4-42
5	c4	2	Select Lockdown TLB Entry for read	WO[e]	-	*TLB lockdown operations* on page 4-43
		4	Select Lockdown TLB Entry for write	WO[e]	-	
	c5	2	Main TLB VA register	RW[e]	-	
	c6	2	Main TLB PA register	RW[e]	-	
	c7	2	Main TLB Attribute register	RW	-	

a. RW in Secure state. Read-only in Non-secure state.

b. Reset value depends on the **MAXCLKLATENCY[2:0]** value. See *Configuration signals* on page A-5.

c. RW in secure privileged mode and RO in Non-secure state and User secure state.

d. In Cortex-A9 uniprocessor implementations the configuration base address is set to zero.

In Cortex-A9 MPCore implementations the configuration base address is reset to **PERIPHBASE[31:13]** so that software can determine

the location of the private memory region.

e. No access in Non-secure state.

4.2.17 Identification Registers

The Processor ID Registers are read-only registers that return the values stored in the Main ID and feature registers of the processor. You must use the CP15 interface to access these registers.

Table 4-16 shows the name, type, value and description that is associated with each Processor ID Register.

Table 4-16 Processor ID Registers

CRn	Op1	CRM	Op2	Name	Type	Value	Description
c0	0	c0	0	MIDR	RO	Product revision dependant	*Main ID Register* on page 4-18
			1	CTR	RO	`0x83338003`	Cache Type Register
			2	TCMTR	RO	`0x00000000`	TCM Type Register
			3	TLBTR[a]	RO	-	*TLB Type Register* on page 4-19
			5	MPIDR	RO	-	*Multiprocessor Affinity Register* on page 4-19
			6	REVIDR	RO	-	*Revision ID register* on page 4-20
		c1	0	ID_PFR0	RO	`0x00001231`	Processor Feature Register 0
			1	ID_PFR1	RO	`0x00000011`	Processor Feature Register 1
			2	ID_DFR0	RO	`0x00010444`	Debug Feature Register 0
			3	ID_AFR0	RO	`0x00000000`	Auxiliary Feature Register 0
			4	ID_MMFR0	RO	`0x00100103`	Memory Model Feature Register 0
			5	ID_MMFR1	RO	`0x20000000`	Memory Model Feature Register 1
			6	ID_MMFR2	RO	`0x01230000`	Memory Model Feature Register 2
			7	ID_MMFR3	RO	`0x00102111`	Memory Model Feature Register 3
		c2	0	ID_ISAR0	RO	`0x00101111`	Instruction Set Attribute Register 0
			1	ID_ISAR1	RO	`0x13112111`	Instruction Set Attribute Register 1
			2	ID_ISAR2	RO	`0x21232041`	Instruction Set Attribute Register 2
			3	ID_ISAR3	RO	`0x11112131`	Instruction Set Attribute Register 3
			4	ID_ISAR4	RO	`0x00011142`	Instruction Set Attribute Register 4
	1	c0	0	CCSIDR	RO	-	*Cache Size Identification Register* on page 4-21
			1	CLIDR	RO	`0x09000003`	*Cache Level ID Register* on page 4-22
			7	AIDR	RO	`0x00000000`	*Auxiliary ID Register* on page 4-23
	2	c0	0	CSSELR	RW	-	*Cache Size Selection Register* on page 4-24

[a] Depends on TLBSIZE. See *TLB Type Register* on page 4-19.

See the *ARM Architecture Reference Manual* for more information on the Processor ID Registers.

4.2.18 Virtual memory control registers

Table 4-17 shows the Virtual memory control registers.

Table 4-17 Virtual memory registers

CRn	Op1	CRm	Op2	Name	Type	Reset	Description
c1	0	c0	0	SCTLR	RW	-[a]	*System Control Register* on page 4-24
c2	0	c0	0	TTBR0	RW	-	
			1	TTBR1	RW	-	Translation Table Base Register 1
			2	TTBCR	RW	`0x00000000`[b]	Translation Table Base Control Register
c3	0	c0	0	DACR	RW	-	Domain Access Control Register
c10	0	c2	0	PRRR	RW	`0x00098AA4`	Primary Region Remap Register
			1	NMRR	RW	`0x44E048E0`	Normal Memory Remap Register
c13	0	c0	1	CONTEXTIDR	RW	-	Context ID Register

a. Depends on input signals. See *System Control Register* on page 4-24.

b. In Secure state only. You must program the Non-secure version with the required value.

4.2.19 Fault handling registers

Table 4-18 shows the Fault handling registers.

Table 4-18 Fault handling registers

CRn	Op1	CRm	Op2	Name	Type	Reset	Description
c5	0	c0	0	DFSR	RW	-	Data Fault Status Register
			1	IFSR	RW	-	Instruction Fault Status Register
		c1	0	ADFSR	-	-	Auxiliary Data Fault Status Register
			1	AIFSR	-	-	Auxiliary Instruction Fault Status Register
c6	0	c0	0	DFAR	RW	-	Data Fault Address Register
			2	IFAR	RW	-	Instruction Fault Address Register

4.2.20 Other system control registers

Table 4-19 shows the other system control registers.

Table 4-19 Other system control registers

CRn	Op1	CRm	Op2	Name	Type	Reset	Description
c1	0	c0	2	CPACR	RW	-[a]	*Coprocessor Access Control Register* on page 4-29

a. The reset value depends on the VFP and NEON configuration. If VFP and NEON are implemented, the reset value is `0x00000000`. If VFP is implemented but NEON is not implemented, the reset value is `0x80000000`. If VFP and NEON are not implemented, the reset value is `0x00000000`.

4.2.21 Cache maintenance operations

Table 4-20 shows the 32-bit wide cache and branch predictor maintenance operations.

Table 4-20 Cache and branch predictor maintenance operations

CRn	Op1	CRm	Op2	Name	Type	Reset	Description
c7	0	c1	0	ICIALLUIS	WO	-	Cache operations registers
			6	BPIALLIS	WO	-	
		c5	0	ICIALLU	WO	-	
			1	ICIMVAU	WO	-	
			6	BPIALL	WO	-	
		c6	1	DCIMVAC	WO	-	
			2	DCISW	WO	-	
		c10	1	DCCVAC	WO	-	
			2	DCCSW	WO	-	
		c11	1	DCCVAU	WO	-	
		c14	1	DCCIMVAC	WO	-	
			2	DCCISW	WO	-	

4.2.22 Address translation operations

Table 4-21 shows the address translation register and operations.

Table 4-21 Address translation operations

CRn	Op1	CRm	Op2	Name	Type	Reset	Description
c7	0	c4	0	PAR	RW	-	-

4.2.23 Miscellaneous operations

Table 4-22 shows the 32-bit wide miscellaneous operations.

Table 4-22 Miscellaneous system control operations

CRn	Op1	CRm	Op2	Name	Type	Reset	Description
c1	0	c1	3	VCR[c]	RW	`0x00000000`	*Virtualization Control Register* on page 4-34
c7	0	c0	4	NOP[a]	WO	-	-
c13	0	c0	2	TPIDRURW	RW[b]	-	Software Thread ID registers
			3	TPIDRURO	RO[c]	-	
			4	TPIDRPRW.	RW	-	

[a] This operation is performed by the `WFI` instruction. See *Deprecated registers* on page 4-2.

[b] RW in User mode.

[c] RO in User mode.

4.2.24 Performance monitor registers

Table 4-23 shows the 32-bit wide performance monitor registers.

Table 4-23 Performance monitor registers

CRn	Op1	CRm	Op2	Name	Type	Reset	Description
c9	0	c12	0	PMCR	RW	`0x41093000`	Performance Monitor Control Register
			1	PMCNTENSET	RW	`0x00000000`	Count Enable Set Register
			2	PMCNTENCLR	RW	`0x00000000`	Count Enable Clear Register
			3	PMOVSR	RW	-	Overflow Flag Status Register
			4	PMSWINC	WO	-	Software Increment Register
			5	PMSELR	RW	`0x00000000`	Event Counter Selection Register
		c13	0	PMCCNTR	RW	-	Cycle Count Register
			1	PMXEVTYPER	RW	-	Event Type Selection Register
			2	PMXEVCNTR	RW	-	Event Count Registers
		c14	0	PMUSERENR	RW[a]	`0x00000000`	User Enable Register
			1	PMINTENSET	RW	`0x00000000`	Interrupt Enable Set Register
			2	PMINTENCLR	RW	`0x00000000`	Interrupt Enable Clear Register

a. RO in User mode.

4.2.25 Security Extensions registers

Table 4-24 shows the Security Extensions registers.

Table 4-24 Security Extensions registers

CRn	Op1	CRm	Op2	Name	Type	Reset	Description
c1	0	c1	0	SCR[a]	RW	`0x00000000`	Secure Configuration Register
			1	SDER[c]	RW	`0x00000000`	*Secure Debug Enable Register* on page 4-31
			2	NSACR	RW[b]	-[c]	*Non-secure Access Control Register* on page 4-32
c12	0	c0	0	VBAR	RW	`0x00000000`[d]	Vector Base Address Register
			1	MVBAR	RW	-	Monitor Vector Base Address Register
		c1	0	ISR	RO	`0x00000000`	Interrupt Status Register

a. No access in Non-secure state.

b. This is a read/write register in Secure state and a read-only register in the Non-secure state.

c. `0x00000000` if NEON present and `0x0000C000` if NEON not present.

d. Only the secure version is reset to 0. The Non-secure version must be programmed by software.

4.2.26 Preload Engine registers

Table 4-25 shows the preload engine registers.

Table 4-25 Preload engine registers

CRn	Op1	CRm	Op2	Name	Type	Reset	Description
11	0	c0	0	PLEIDR	RO[a]	-	*PLE ID Register* on page 4-36
			2	PLEASR	RO[a]	-	*PLE Activity Status Register* on page 4-36
			4	PLEFSR	RO[a]	-	*PLE FIFO Status Register* on page 4-37
		c1	0	PLEUAR	Privileged R/W User RO	-	*Preload Engine User Accessibility Register* on page 4-38
			1	PLEPCR	Privileged R/W User RO	-	*Preload Engine Parameters Control Register* on page 4-39

a. RAZ if the PLE is not present.

4.2.27 TLB maintenance

Table 4-25 shows the TLB maintenance operations and registers.

Table 4-26 TLB **maintenance**

CRn	Op1	CRm	Op2	Name	Type	Reset	Description
c8	0	c3	0	TLBIALLIS[a]	WO	-	-
			1	TLBIMVAIS[b]	WO	-	-
			2	TLBIASIDIS[b]	WO	-	-
			3	TLBIMVAAIS[a]	WO	-	-
		c5, c6, or c7	0	TLBIALL[a]	WO	-	-
			1	TLBIMVA[b]	WO	-	-
			2	TLBIASID[b]	WO	-	-
			3	TLBIMVAA[a]	WO	-	-
c10	0	c0	0	TLB Lockdown Register[c]	RW	`0x00000000`	*TLB Lockdown Register* on page 4-35
c15	5	c4	2	Select Lockdown TLB Entry for read	WO[d]	-	*TLB lockdown operations* on page 4-43
			4	Select Lockdown TLB Entry for write	WO[e]	-	
		c5	2	Main TLB VA register	RW[e]	-	
		c6	2	Main TLB PA register	RW[e]	-	
		c7	2	Main TLB Attribute register	RW	-	

a. Has no effect on entries that are locked down.

b. Invalidates the locked entry when it matches.

c. No access in Non-secure state if NSCAR.TL=0 and RW if NSACR.TL=1.

d. No access in Non-secure state.

4.2.28 Implementation defined registers

Table 4-27 shows the implementation defined registers. These registers provide test features and any required configuration options specific to the Cortex-A9 processor.

Table 4-27 Implementation defined registers

CRn	Op1	CRm	Op2	Name	Type	Reset	Description
c1	0	c0	1	ACTLR[a]	RW	`0x00000000`	*Auxiliary Control Register* on page 4-27
	4	c0	0	Configuration Base Address	RO[b]	- [c]	*Configuration Base Address Register* on page 4-42

---

a. RO in Non-secure state if NSACR[18]=0 and RW if NSACR[18]=1.

b. RW in secure privileged mode and RO in Non-secure state and User secure state.

c. In Cortex-A9 uniprocessor implementations the configuration base address is set to zero.

In Cortex-A9 MPCore implementations the configuration base address is reset to **PERIPHBASE[31:13]** so that software can determine

the location of the private memory region.

4.3 Register descriptions

This section describes the implementation-defined CP15 system control registers by coprocessor register number order that are not already described in the *ARM Architecture Reference Manual*.

4.3.1 Main ID Register

The MIDR characteristics are:

Purpose	Provides identification information for the processor, including an implementer code for the device and a device ID number.
Usage constraints	The MIDR is: • a read-only register • common to the Secure and Non-secure states • only accessible in privileged modes.
Configurations	Available in all configurations.
Attributes	See the register summary in Table 4-2 on page 4-5.

Figure 4-1 shows the MIDR bit assignments.

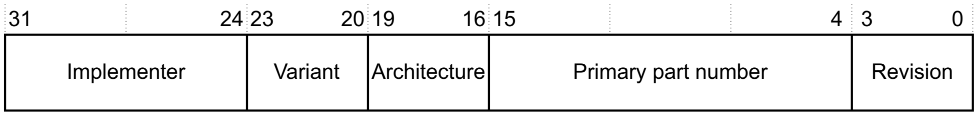


Figure 4-1 MIDR bit assignments

Table 4-28 shows the MIDR bit assignments.

Table 4-28 MIDR bit assignments

Bits	Name	Function
[31:24]	Implementer	Indicates the implementer code: `0x41` ARM Limited.
[23:20]	Variant	Indicates the variant number of the processor. This is the major revision number *n* in the *rn* part of the *rnpn* description of the product revision status, for example: `0x3` Major revision number.
[19:16]	Architecture	Indicates the architecture code: `0xF` Defined by CPUID scheme.
[15:4]	Primary part number	Indicates the primary part number: `0xC09` Cortex-A9.
[3:0]	Revision	Indicates the minor revision number of the processor. This is the minor revision number *n* in the *pn* part of the *rnpn* description of the product revision status, for example: `0x0` Minor revision number.

To access the MIDR, read the CP15 register with:

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

4.3.2 TLB Type Register

The TLBTR characteristics are:

Purpose Returns the number of lockable entries for the TLB.

Usage constraints The TLBTR is:

- common to the Secure and Non-secure states
- only accessible in privileged mode.

Configurations Available in all configurations.

Attributes See the register summary in Table 4-2 on page 4-5.

Figure 4-2 shows the TLBTR bit assignments.

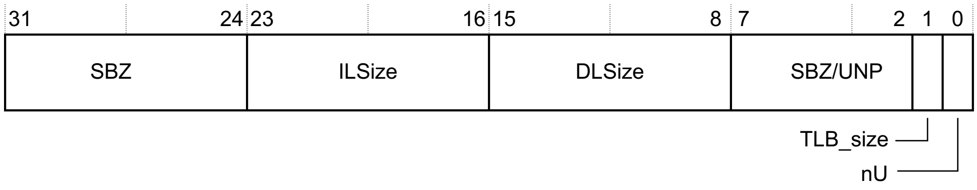


Figure 4-2 TLBTR bit assignments

Table 4-29 shows the TLBTR bit assignments.

Table 4-29 TLBTR bit assignments

Bits	Name	Function
[31:24]	SBZ	-
[23:16]	ILsize	Specifies the number of instruction TLB lockable entries. For the Cortex-A9 processor this is 0.
[15:8]	DLsize	Specifies the number of unified or data TLB lockable entries. For the Cortex-A9 processor this is 4.
[7:2]	SBZ or UNP	-
[1]	TLB_size	0 = TLB has 64 entries 1 = TLB has 128 entries.
[0]	nU	Specifies if the TLB is unified, 0, or if there are separate instruction and data TLBs. 0 = The Cortex-A9 processor has a unified TLB.

To access the TLBTR, read the CP15 register with:

```
MRC p15,0,<Rd>,c0,c0,3; returns TLB details
```

4.3.3 Multiprocessor Affinity Register

The MPIDR characteristics are:

Purpose To identify:

- whether the processor is part of a Cortex-A9 MPCore implementation
- Cortex-A9 processor accesses within a Cortex-A9 MPCore processor
- the target Cortex-A9 processor in a multi-processor cluster system.

Usage constraints The MPIDR is:

- only accessible in privileged mode
- common to the Secure and Non-secure states.

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

Attributes See the register summary in Table 4-2 on page 4-5.

Figure 4-3 shows the MPIDR bit assignments.

Figure 4-3 MPIDR bit assignments

Table 4-30 shows the MPIDR bit assignments.

Table 4-30 MPIDR bit assignments

Bits	Name	Function
[31]	-	Indicates the register uses the new multiprocessor format. This is always 1.
[30]	U bit	Multiprocessing Extensions: 0 = processor is part of an MPCore cluster 1 = processor is a uniprocessor.
[29:12]	-	SBZ.
[11:8]	Cluster ID	Value read in **CLUSTERID** configuration pins[a]. It identifies a Cortex-A9 MPCore processor in a system with more than one Cortex-A9 MPCore processor present. SBZ for a uniprocessor configuration.
[7:2]	-	SBZ.
[1:0]	CPU ID	Indicates the CPU number in the Cortex-A9 MPCore configuration: `0x0` = processor is CPU0 `0x1` = processor is CPU1 `0x2` = processor is CPU2 `0x3` = processor is CPU3. In the uniprocessor version this value is fixed at `0x0`.

[a] A uniprocessor implementation does not include any **CLUSTERID** pins.

To access the MPIDR, read the CP15 register with:

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

4.3.4 Revision ID register

The REVIDR characteristics are:

Purpose Provides implementation-specific minor revision information that can only be interpreted in conjunction with the MIDR.

Usage constraints The REVIDR is:

- a read-only register
- common to the Secure and Non-secure states
- only accessible in privileged modes.

Configurations Available in all configurations.

Attributes See the register summary in Table 4-2 on page 4-5.

Figure 4-4 shows the REVIDR bit assignments.

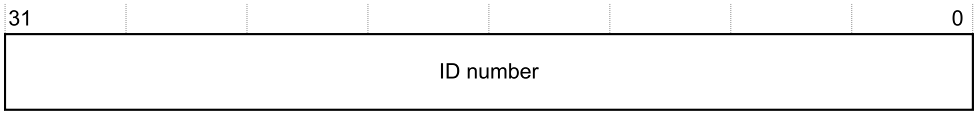


Figure 4-4 REVIDR bit assignments

Table 4-31 shows the REVIDR bit assignments.

Table 4-31 REVIDR bit assignments

Bits	Name	Function
[31:0]	ID number	Implementation-specific revision information. The reset value is determined by the specific Cortex-A9 implementation.

To access the REVIDR, read the CP15 register with:

```
MRC p15, 0, <Rt>, c0, c0, 6; Read Revision ID Register
```

4.3.5 Cache Size Identification Register

The CCSIDR characteristics are:

Purpose Provides information about the architecture of the caches selected by CSSELR.

Usage constraints The CCSIDR is:

- only accessible in privileged modes
- common to the Secure and Non-secure states.

Configurations Available in all configurations.

Attributes See the register summary in Table 4-2 on page 4-5.

Figure 4-5 shows the CCSIDR bit assignments.

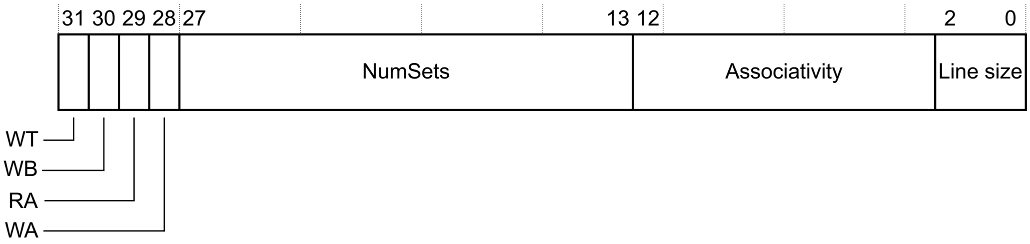


Figure 4-5 CCSIDR bit assignments

Table 4-32 shows how the CSSIDR bit assignments.

Table 4-32 CCSIDR bit assignments

Bits	Name	Function
[31]	WT	Indicates support available for Write-Through: 0 = Write-Through support not available 1 = Write-Through support available.
[30]	WB	Indicates support available for Write-Back: 0 = Write-Back support not available 1 = Write-Back support available.
[29]	RA	Indicates support available for Read-Allocation: 0 = Read-Allocation support not available 1 = Read-Allocation support available.
[28]	WA	Indicates support available for Write-Allocation: 0 = Write-Allocation support not available 1 = Write-Allocation support available.
[27:13]	NumSets	Indicates number of sets. `0x7F` = 16KB cache size `0xFF` = 32KB cache size `0x1FF` = 64KB cache size.
[12:3]	Associativity	Indicates number of ways. b0000000011 = four ways.
[2:0]	LineSize	Indicates number of words. b001 = 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, then bits [31:28] are b0010.

If the CSSELR reads the data cache values, then bits [31:28] are b0111. See *Cache Size Selection Register* on page 4-24.

4.3.6 Cache Level ID Register

The CLIDR characteristics are:

Purpose Identifies:

- the type of cache, or caches, implemented at each level
- the Level of Coherency and Level of Unification for the cache hierarchy.

Usage constraints The CLIDR is:

- only accessible in privileged modes
- common to the Secure and Non-secure states.

Configurations Available in all configurations.

Attributes See the register summary in Table 4-2 on page 4-5.

Figure 4-6 shows the CLIDR bit assignments.

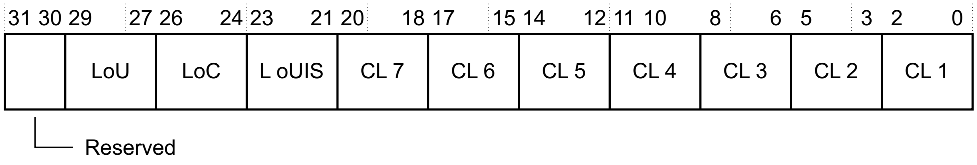


Figure 4-6 CLIDR bit assignments

Table 4-33 shows the CLIDR bit assignments.

Table 4-33 CLIDR bit assignments

Bits	Name	Function
[31:30]	-	UNP or SBZ
[29:27]	LoU	b001 = level of unification
[26:24]	LoC	b001 = level of coherency
[23:21]	LoUIS	b001 = level of Unification Inner Shareable
[20:18]	CL 7	b000 = no cache at CL 7
[17:15]	CL 6	b000 = no cache at CL 6
[14:12]	CL 5	b000 = no cache at CL 5
[11:9]	CL 4	b000 = no cache at CL 4
[8:6]	CL 3	b000 = no cache at CL 3
[5:3]	CL 2	b000 = no unified cache at CL 2
[2:0]	CL 1	b011 = separate instruction and data caches at CL 1

To access the CLIDR, read the CP15 register with:

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

4.3.7 Auxiliary ID Register

The AIDR characteristics are:

Purpose	Provides implementation-specific information.
Usage constraints	The AIDR is: • only accessible in privileged modes • common to the Secure and Non-secure states.
Configurations	Available in all configurations.
Attributes	See the register summary in Table 4-2 on page 4-5.

To access the Auxiliary Level ID Register, read the CP15 register with:

```
MRC p15,1,<Rd>,c0,c0,7; Read Auxiliary ID Register
```

Note

The AIDR is unused in this implementation.

4.3.8 Cache Size Selection Register

The CSSELR characteristics are:

Purpose Selects the current CCSIDR. See the *Cache Size Identification Register* on page 4-21.

Usage constraints The CSSELR is:

- only accessible in privileged modes
- banked for Secure and Non-secure states.

Configurations Available in all configurations.

Attributes See the register summary in Table 4-2 on page 4-5.

Figure 4-7 shows the CSSELR bit assignments.

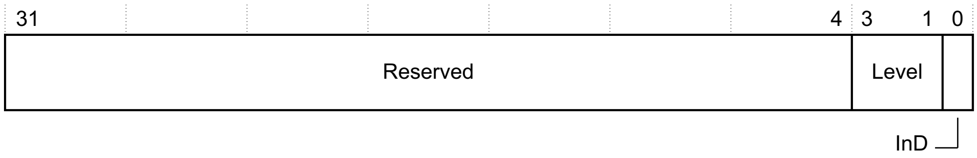


Figure 4-7 CSSELR bit assignments

Table 4-34 shows the CSSELR bit assignments.

Table 4-34 CSSELR bit assignments

Bits	Name	Function
[31:4]	-	UNP or SBZ.
[3:1]	Level	Cache level selected, RAZ/WI. There is only one level of cache in the Cortex-A9 processor so the value for this field is b000.
[0]	InD	Instruction not Data bit: 0 = data cache 1 = instruction cache.

To access the CSSELR, read the CP15 register with:

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

4.3.9 System Control Register

The SCTLR characteristics are:

Purpose Provides control and configuration of:

- memory alignment and endianness
- memory protection and fault behavior
- MMU and cache enables
- interrupts and behavior of interrupt latency
- location for exception vectors

- program flow prediction.

Usage constraints The SCTLR is:

- Only accessible in privileged modes.
- Partially banked. Table 4-35 shows banked and secure modify only bits.

Configurations Available in all configurations.

Attributes See the register summary in Table 4-3 on page 4-6.

Figure 4-8 shows the SCTLR bit assignments.

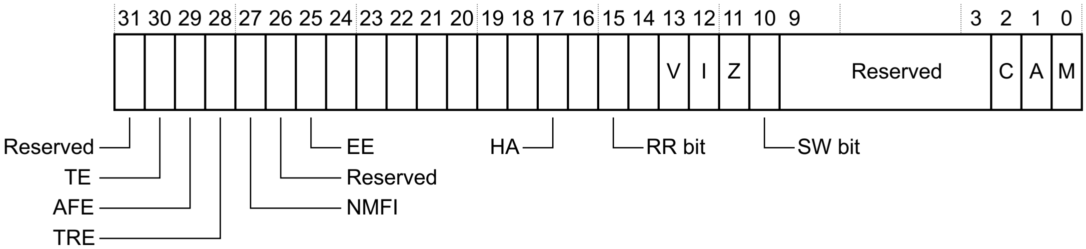


Figure 4-8 SCTLR bit assignments

Table 4-35 shows the SCTLR bit assignments.

Table 4-35 SCTLR bit assignments

Bits	Name	Access	Function
[31]	-	-	SBZ.
[30]	TE	Banked	Thumb exception enable: 0 = exceptions including reset are handled in ARM state 1 = exceptions including reset are handled in Thumb state. The **TEINIT** signal defines the reset value.
[29]	AFE	Banked	Access Flag enable bit: 0 = Full access permissions behavior. This is the reset value. The software maintains binary compatibility with ARMv6K behavior. 1 = Simplified access permissions behavior. The Cortex-A9 processor redefines the AP[0] bit as an access flag. The TLB must be invalidated after changing the AFE bit.
[28]	TRE	Banked	This bit controls the TEX remap functionality in the MMU: 0 = TEX remap disabled. This is the reset value. 1 = TEX remap enabled.
[27]	NMFI	Read-only	Non-maskable FIQ support. The bit cannot be configured by software. The **CFGNMFI** signal defines the reset value.
[26]	-	-	RAZ/SBZP.

Table 4-35 SCTLR bit assignments (continued)

Bits	Name	Access	Function
[25]	EE bit	Banked	Determines how the E bit in the CPSR is set on an exception: 0 = CPSR E bit is set to 0 on an exception 1 = CPSR E bit is set to 1 on an exception. This value also indicates the endianness of the translation table data for translation table lookups. 0 = little-endian 1 = big-endian. The **CFGEND** signal defines the reset value.
[24]	-	-	RAZ/WI.
[23:22]	-	-	RAO/SBOP.
[21]	-	-	RAZ/WI.
[20:19]	-	-	RAZ/SBZP.
[18]	-	-	RAO/SBOP.
[17]	HA	-	RAZ/WI.
[16]	-	-	RAO/SBOP.
[15]	-	-	RAZ/SBZP.
[14]	RR	Secure modify only	Replacement strategy for the instruction cache, the BTAC, and the instruction and data micro TLBs. This bit is read/write in Secure state and read-only in Non-secure state: 0 = Random replacement. This is the reset value. 1 = Round-robin replacement.
[13]	V	Banked	Vectors bit. This bit selects the base address of the exception vectors: 0 = Normal exception vectors, base address `0x00000000`. The Security Extensions are implemented, so this base address can be remapped. 1 = High exception vectors, Hivecs, base address `0xFFFF0000`. This base address is never remapped. At reset the value for the secure version if this bit is taken from **VINITHI**.
[12]	I bit	Banked	Determines if instructions can be cached at any available cache level: 0 = Instruction caching disabled at all levels. This is the reset value. 1 = Instruction caching enabled.
[11]	Z bit	Banked	Enables program flow prediction: 0 = Program flow prediction disabled. This is the reset value. 1 = Program flow prediction enabled.
[10]	SW bit	Banked	SWP/SWPB enable bit: 0 = SWP and SWPB are UNDEFINED.This is the reset value. 1 = SWP and SWPB perform normally.
[9:7]	-	-	RAZ/SBZP.
[6:3]	-	-	RAO/SBOP.

Table 4-35 SCTLR bit assignments (continued)

Bits	Name	Access	Function
[2]	C bit	Banked	Determines if data can be cached at any available cache level: 0 = Data caching disabled at all levels. This is the reset value. 1 = Data caching enabled.
[1]	A bit	Banked	Enables strict alignment of data to detect alignment faults in data accesses: 0 = Strict alignment fault checking disabled. This is the reset value. 1 = Strict alignment fault checking enabled.
[0]	M bit	Banked	Enables the MMU: 0 = MMU disabled. This is the reset value. 1 = MMU enabled.

Attempts to read or write the SCTLR from secure or Non-secure User modes result in an Undefined Instruction exception.

Attempts to write to this register in secure privileged mode when **CP15SDISABLE** is HIGH result in an Undefined Instruction exception.

Attempts to write secure modify only bits in non-secure privileged modes are ignored.

Attempts to read secure modify only bits return the secure bit value.

Attempts to modify read-only bits are ignored.

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

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

4.3.10 Auxiliary Control Register

The ACTLR characteristics are:

Purpose Controls:

- parity checking, if implemented
- allocation in one way
- exclusive caching with the L2 cache
- coherency mode, *Symmetric Multiprocessing* (SMP) or *Asymmetric Multiprocessing* (AMP)
- speculative accesses on AXI
- broadcast of cache, branch predictor, and TLB maintenance operations
- write full line of zeros mode optimization for L2C-310 cache requests.

Usage constraints The ACTLR is:

- Only accessible in privileged modes.
- Common to the Secure and Non-secure states.
- RW in Secure state.
- RO in Non-secure state if NSACR.NS_SMP = 0.

- RW in Non-secure state if NSACR.NS_SMP = 1. In this case all bits are Write Ignore except for the SMP bit.

Configurations Available in all configurations.

- In all configurations when the SMP bit = 0, Inner Cacheable Shareable attributes are treated as Non-cacheable.
- In multiprocessor configurations when the SMP bit is set:
  - — broadcasting cache and TLB maintenance operations is permitted if the FW bit is set
  - — receiving cache and TLB maintenance operations broadcast by other Cortex-A9 processors in the same coherent cluster is permitted if the FW bit is set
  - — the Cortex-A9 processor can send and receive coherent requests for Shared Inner Write-back Write-Allocate accesses from other Cortex-A9 processors in the same coherent cluster.

Attributes See the register summary in Table 4-3 on page 4-6.

Figure 4-9 shows the ACTLR bit assignments.

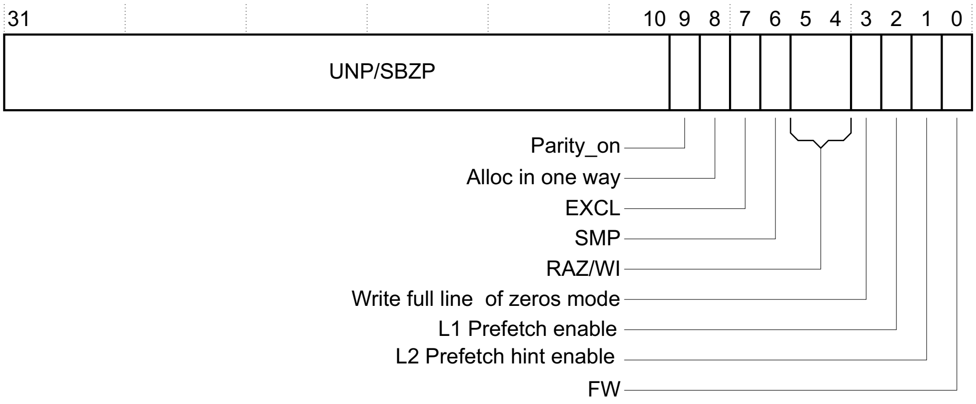


Figure 4-9 ACTLR bit assignments

Table 4-36 shows the ACTLR bit assignments.

Table 4-36 ACTLR bit assignments

Bits	Name	Function
[31:10]	-	UNP or SBZP.
[9]	Parity on	Support for parity checking, if implemented: 0 = Disabled. This is the reset value. 1 = Enabled. If parity checking is not implemented this bit reads as zero and writes are ignored.
[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.

Table 4-36 ACTLR bit assignments (continued)

Bits	Name	Function
[7]	EXCL	Exclusive cache bit. The exclusive cache configuration does not permit data to reside in L1 and L2 at the same time. The exclusive cache configuration provides support for only caching data on an eviction from L1 when the inner cache attributes are Write-Back, Cacheable and allocated in L1. Ensure that your cache controller is also configured for exclusive caching. 0 = Disabled. This is the reset value. 1 = Enabled.
[6]	SMP	Signals if the Cortex-A9 processor is taking part in coherency or not. In uniprocessor configurations, if this bit is set, then Inner Cacheable Shared is treated as Cacheable. The reset value is zero.
[5:4]	-	RAZ/WI.
[3]	Write full line of zeros mode	Enable write full line of zeros mode[a]. The reset value is zero.
[2]	L1 prefetch enable	Dside prefetch. 0 = Disabled. This is the reset value. 1 = Enabled.
[1]	L2 prefetch enable	Prefetch hint enable[a]. The reset value is zero.
[0]	FW	Cache and TLB maintenance broadcast: 0 = Disabled. This is the reset value. 1 = Enabled. RAZ/WI if only one Cortex-A9 processor is present.

[a] This feature must be enabled only when the slaves connected on the Cortex-A9 AXI master port support it. The L2-310 Cache Controller supports this feature. See *Optimized accesses to the L2 memory interface* on page 8-7.

To access the ACTLR you must use a read modify write technique. 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
```

Attempts to write to this register in secure privileged mode when **CP15SDISABLE** is HIGH result in an Undefined Instruction exception.

4.3.11 Coprocessor Access Control Register

The CPACR characteristics are:

Purpose

- sets access rights for the coprocessors CP11 and CP10
- enables software to determine if any particular coprocessor exists in the system.

Note

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

Usage constraints The CPACR is:

- only accessible in privileged modes

- common to Secure and Non-secure states.

Configurations Available in all configurations.

Attributes See the register summary in Table 4-3 on page 4-6.

Figure 4-10 shows the CPACR bit assignments.

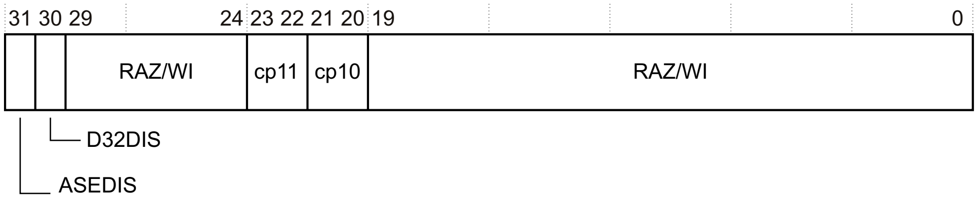


Figure 4-10 CPACR bit assignments

Table 4-37 shows the CPACR bit assignments.

Table 4-37 CPACR bit assignments

Bits	Name	Function
[31]	ASEDIS	Disable Advanced SIMD Extension functionality: 0 = all Advanced SIMD and VFP instructions execute normally 1 = all Advanced SIMD instructions that are not VFP instructions are UNDEFINED. See the *Cortex-A9 Floating-Point Unit Technical Reference Manual* and *Cortex-A9 NEON Media Processing Engine Technical Reference Manual* for more information. If implemented with VFP only, this bit is RAO/WI. If implemented without both VFP and NEON, this bit is UNK/SBZP.
[30]	D32DIS	Disable use of D16-D31 of the VFP register file: 0 = all VFP instructions execute normally 1 = all VFP instructions are UNDEFINED if they access any of registers D16-D31. See the *Cortex-A9 Floating-Point Unit Technical Reference Manual* and *Cortex-A9 NEON Media Processing Engine Technical Reference Manual* for more information. If implemented with VFP only, this bit is RAO/WI. If implemented without both VFP and NEON, this bit is UNK/SBZP.
[29:24]	-	RAZ/WI.
[23:22]	cp11	Defines access permissions for the coprocessor. Access denied is the reset condition and is the behavior for non-existent coprocessors. b00 = Access denied. This is the reset value. Attempted access generates an Undefined Instruction exception. b01 = Privileged mode access only. b10 = Reserved. b11 = Privileged and User mode access.
[21:20]	cp10	Defines access permissions for the coprocessor. Access denied is the reset condition and is the behavior for non-existent coprocessors. b00 = Access denied. This is the reset value. Attempted access generates an Undefined Instruction exception. b01 = Privileged mode access only. b10 = Reserved. b11 = Privileged and User mode access.
[19:0]	-	RAZ/WI.

Access to coprocessors in the Non-secure state depends on the permissions set in the *Non-secure Access Control Register* on page 4-32.

Attempts to read or write the CPACR access bits depend on the corresponding bit for each coprocessor in *Non-secure Access Control Register* on page 4-32.

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 the *ARM Architecture Reference Manual* for more information. 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 b11. If the coprocessor does not exist in the system the access rights remain set to b00.

Note

You must enable both coprocessor 10 and coprocessor 11 before accessing any NEON or VFP system registers.

4.3.12 Secure Debug Enable Register

The SDER characteristics are:

Purpose	Controls Cortex-A9 debug.
Usage constraints	The SDER is: • only accessible in privileged modes • only accessible in Secure state, accesses in Non-secure state cause an Undefined Instruction exception.
Configurations	Available in all configurations.
Attributes	See the register summary in Table 4-3 on page 4-6.

Figure 4-11 shows the SDER bit assignments.

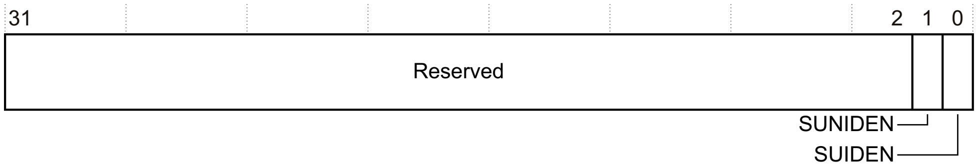


Figure 4-11 SDER bit assignments

Table 4-38 shows the SDER bit assignments.

Table 4-38 SDER bit assignments

Bits	Name	Function
[31:2]	-	Reserved.
[1]	Secure User Non-invasive Debug Enable	0 = Non-invasive debug not permitted in Secure User mode. This is the reset value. 1 = Non-invasive debug permitted in Secure User mode.
[0]	Secure User Invasive Debug Enable	0 = Invasive debug not permitted in Secure User mode. This is the reset value. 1 = Invasive debug permitted in Secure User mode.

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

```
MRC p15,0,<Rd>,c1,c1,1; Read Secure debug enable Register
MCR p15,0,<Rd>,c1,c1,1; Write Secure debug enable Register
```

4.3.13 Non-secure Access Control Register

The NSACR characteristics are:

Purpose Sets the Non-secure access permission for coprocessors.

Usage constraints The NSACR is:

- only accessible in privileged modes
- a read/write register in Secure state
- a read-only register in Non-secure state.

Note

This register has no effect on Non-secure access permissions for the debug control coprocessor, or the system control coprocessor.

Configurations Available in all configurations.

Attributes See the register summary in Table 4-3 on page 4-6.

Figure 4-12 shows the NSACR bit assignments.

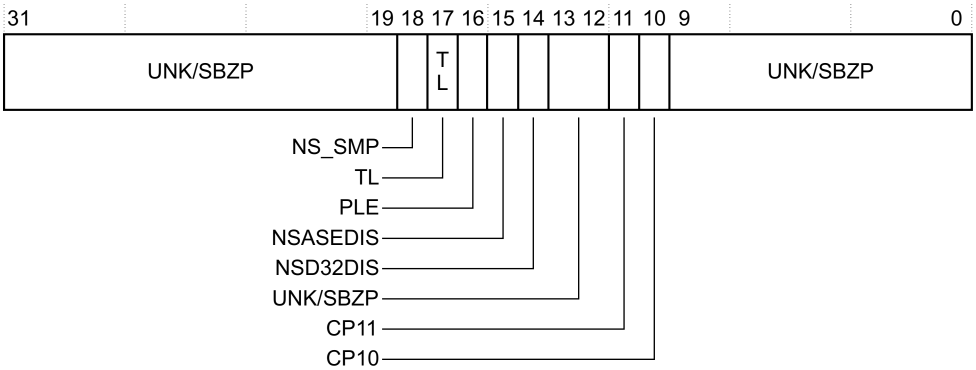


Figure 4-12 NSACR bit assignments

Table 4-39 shows the NSACR bit assignments.

Table 4-39 NSACR bit assignments

Bits	Name	Function
[31:19]	-	UNK/SBZP.
[18]	NS_SMP	Determines if the SMP bit of the Auxiliary Control Register is writable in Non-secure state: 0 = A write to Auxiliary Control Register in Non-secure state takes an Undefined Instruction exception and the SMP bit is write ignored. This is the reset value. 1= A write to Auxiliary Control Register in Non-secure state can modify the value of the SMP bit. Other bits are write ignored.
[17]	TL	Determines if lockable TLB entries can be allocated in Non-secure state: 0 = Lockable TLB entries cannot be allocated. This is the reset value. 1 = Lockable TLB entries can be allocated.
[16]	PLE	Controls NS accesses to the Preload Engine resources: 0 = Only Secure accesses to CP15 c11 are permitted. All Non-secure accesses to CP15 c11 are trapped to UNDEFINED. This is the default value. 1 = Non-secure accesses to the CP15 c11 domain are permitted. That is, PLE resources are available in the Non-secure state. If the Preload Engine is not implemented, this bit is RAZ/WI. See Chapter 9 *Preload Engine*.
[15]	NSASEDIS	Disable Non-secure Advanced SIMD Extension functionality: 0 = This bit has no effect on the ability to write CPACR.ASEDIS. This is the reset value. 1 = The CPACR.ASEDIS bit when executing in Non-secure state has a fixed value of 1 and writes to it are ignored. See the *Cortex-A9 Floating-Point Unit Technical Reference Manual* and *Cortex-A9 NEON Media Processing Engine Technical Reference Manual* for more information.
[14]	NSD32DIS	Disable the Non-secure use of D16-D31 of the VFP register file: 0 = This bit has no effect on the ability to write CPACR. D32DIS. This is the reset value. 1 = The CPACR.D32DIS bit when executing in Non-secure state has a fixed value of 1 and writes to it are ignored. See the *Cortex-A9 Floating-Point Unit Technical Reference Manual* and *Cortex-A9 NEON Media Processing Engine Technical Reference Manual* for more information.
[13:12]	-	UNK/SBZP.
[11]	CP11	Determines permission to access coprocessor 11 in the Non-secure state: 0 = Secure access only. This is the reset value. 1 = Secure or Non-secure access.
[10]	CP10	Determines permission to access coprocessor 10 in the Non-secure state: 0 = Secure access only. This is the reset value. 1 = Secure or Non-secure access.
[9:0]	-	UNK/SBZP.

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

```
MRC p15, 0,<Rd>, c1, c1, 2; Read NSACR data
MCR p15, 0,<Rd>, c1, c1, 2; Write NSACR data
```

See the *Cortex-A9 Floating-Point Unit Technical Reference Manual* and *Cortex-A9 NEON Media Processing Engine Technical Reference Manual* for more information.

4.3.14 Virtualization Control Register

The VCR characteristics are:

Purpose	Forces an exception regardless of the value of the A, I, or F bits in the *Current Program Status Register* (CPSR).
Usage constraints	The VCR is: • only accessible in privileged modes • only accessible in Secure state.
Configurations	Available in all configurations.
Attributes	See the register summary in Table 4-3 on page 4-6.

Figure 4-13 shows the VCR bit assignments.

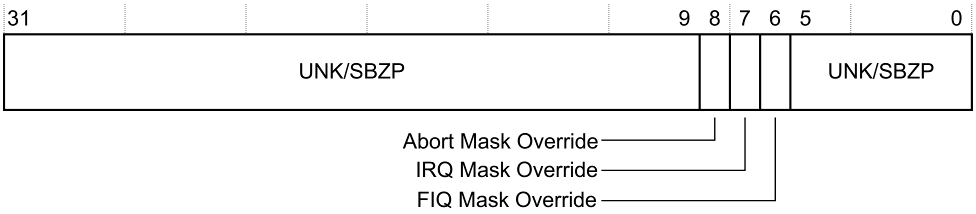


Figure 4-13 VCR bit assignments

Table 4-40 shows the VCR bit assignments.

Table 4-40 VCR bit assignments

Bits	Name	Function
[31:9]	-	UNK/SBZP.
[8]	AMO	Abort Mask Override. When the processor is in Non-secure state and the SCR.EA bit is set, if the AMO bit is set, this enables an asynchronous Data Abort exception to be taken regardless of the value of the CPSR.A bit. When the processor is in Secure state, or when the SCR.EA bit is not set, the AMO bit is ignored.
[7]	IMO	IRQ Mask Override. When the processor is in Non-secure state and the SCR.IRQ bit is set, if the IMO bit is set, this enables an IRQ exception to be taken regardless of the value of the CPSR.I bit. When the processor is in Secure state, or when the SCR.IRQ bit is not set, the IMO bit is ignored.
[6]	IFO	FIQ Mask Override. When the processor is in Non-secure state and the SCR.FIQ bit is set, if the IFO bit is set, this enables an FIQ exception to be taken regardless of the value of the CPSR.F bit. When the processor is in Secure state, or when the SCR.FIQ bit is not set, the IFO bit is ignored.
[5:0]	-	UNK/SBZP.

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

```
MRC p15, 0,<Rd>, c1, c1, 3; Read VCR data
MCR p15, 0,<Rd>, c1, c1, 3; Write VCR data
```

4.3.15 TLB Lockdown Register

The TLB Lockdown Register characteristics are:

Purpose Controls where hardware translation table walks place the TLB entry. The TLB entry can be in either:

- The set-associative region of the TLB.
- The lockdown region of the TLB, and if in the lockdown region, the entry to write.

  The lockdown region of the TLB contains four entries.

Usage constraints The TLB Lockdown Register is:

- only accessible in privileged modes
- common to Secure and Non-secure states
- not accessible if NSACR.TL is 0.

Configurations Available in all configurations.

Attributes See the register summary in Table 4-11 on page 4-9.

Figure 4-14 shows the TLB Lockdown Register bit assignments.

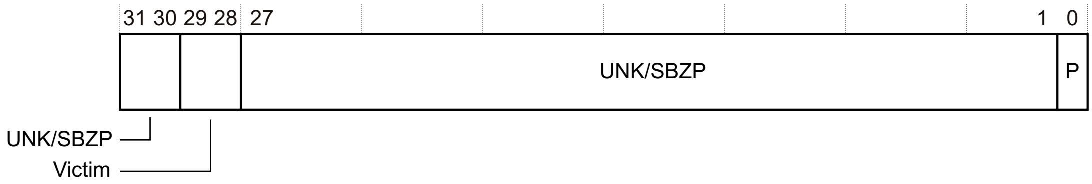


Figure 4-14 TLB Lockdown Register bit assignments

Table 4-41 shows the TLB Lockdown Register bit assignments

Table 4-41 TLB Lockdown Register bit assignments

Bits	Name	Function
[31:30]	-	UNK/SBZP.
[29:28]	Victim	Lockdown region.
[27:1]	-	UNK/SBZP.
[0]	P	Preserve bit. The reset value is `0`.

To access the TLB Lockdown Register, read or write the CP15 register with:

```
MRC p15, 0,<Rd>, c10, c0, 0; Read TLB Lockdown victim
MCR p15, 0,<Rd>, c10, c0, 0; Write TLB Lockdown victim
```

Writing the TLB Lockdown Register with the preserve bit (P bit) set to:

1 Means subsequent hardware translation table walks place the TLB entry in the lockdown region at the entry specified by the victim, in the range 0 to 3.

0 Means subsequent hardware translation table walks place the TLB entry in the set-associative region of the TLB.

See *Invalidate TLB Entries on ASID Match* on page 4-45.

4.3.16 PLE ID Register

The PLEIDR characteristics are:

Purpose Indicates whether the PLE is present or not and the size of its FIFO.

Usage constraints The PLEIDR is:

- common to Secure and Non-secure states
- accessible in User and privileged modes, regardless of any configuration bit.

Configurations Available in all Cortex-A9 configurations regardless of whether a PLE is present or not.

Attributes See Table 4-12 on page 4-10.

Figure 4-15 shows the PLEIDR bit assignments.

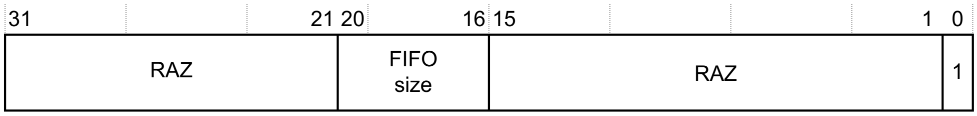


Figure 4-15 PLEIDR bit assignments

Table 4-42 shows the PLEIDR bit assignments.

Table 4-42 PLEIDR bit assignments

Bits	Name	Function
[31:21]	-	-
[20:16]	PLE FIFO size	Permitted values are: • 5'b00000 indicates the PLE is not present • 5'b00100 indicates a PLE is present with a FIFO size of 4 entries • 5'b01000 indicates a PLE is present with a FIFO size of 8 entries • 5'b10000 indicates a PLE is present with a FIFO size of 16 entries.
[15:1]	-	RAZ.
[0]	-	A value of 1 indicates that the Preload Engine is present in the given configuration.

To access the PLEIDR, read the CP15 register with:

```
MRC p15, 0, <Rt>, c11, c0, 0; Read PLEIDR
```

4.3.17 PLE Activity Status Register

The PLEASR characteristics are:

Purpose Indicates whether the PLE engine is active.

Usage constraints The PLEASR is:

- common to Secure and Non-secure states
- accessible in User and privileged modes, regardless of any configuration bit.

Configurations Available in all Cortex-A9 configurations regardless of whether a PLE is present or not.

Attributes See Table 4-12 on page 4-10.

Figure 4-16 shows the PLEASR bit assignments.

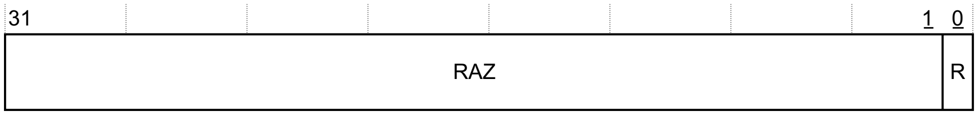


Figure 4-16 PLEASR bit assignments

Table 4-43 shows the PLEASR bit assignments.

Table 4-43 PLEASR bit assignments

Bits	Name	Function
[31:1]	-	Reserved, RAZ
[0]	R	PLE Channel running: 1 = the Preload Engine is handling a PLE request.

To access the PLEASR, read the CP15 register with:

```
MRC p15, 0, <Rt>, c11, c0, 2; Read PLEASR
```

4.3.18 PLE FIFO Status Register

The PLEFSR characteristics are:

Purpose Indicates how many entries remain available in the PLE FIFO.

Usage constraints The PLEFSR is:

- common to Secure and Non-secure states
- accessible in User and privileged modes, regardless of any configuration bit.

NSAC.PLE controls Non-secure accesses.

Configurations Available in all Cortex-A9 configurations regardless of whether a PLE is present or not.

Attributes See Table 4-12 on page 4-10.

Figure 4-17 shows the PLEFSR bit assignments.

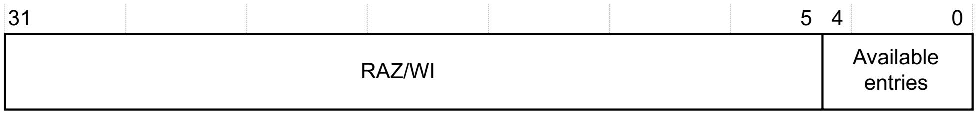


Figure 4-17 PLESFR bit assignments

Table 4-44 shows the PLEFSR bit assignments.

Table 4-44 PLESFR bit assignments

Bits	Name	Function
[31:5]	-	Reserved, RAZ/WI.
[4:0]	Available entries	Number of available entries in the PLE FIFO. This is the difference between the total number of entries in the FIFO, that is configuration-specific, and the number of entries already programmed.

Use the PLESFR to check that an entry is available before programming a new PLE channel.

To access the PLESFR, read the CP15 register with:

```
MRC p15, 0, <Rt>, c11, c0, 4; Read the PLESFR
```

4.3.19 Preload Engine User Accessibility Register

The PLEUAR characteristics are:

Purpose Controls whether PLE operations are available in User mode.

Usage constraints The PLEUAR is:

- common to Secure and Non-secure states
- accessible in User and privileged modes, regardless of any configuration bit.

Configurations Only available in configurations where the Preload Engine is present, otherwise an Undefined Instruction exception is taken.

Attributes See Table 4-12 on page 4-10.

Figure 4-18 shows the PLEUAR bit assignments.

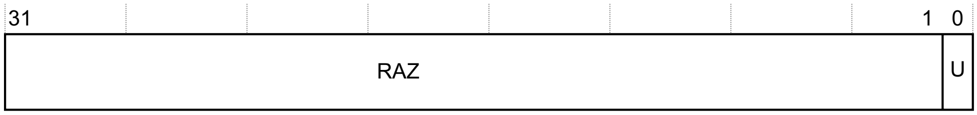


Figure 4-18 PLEUAR bit assignments

Table 4-45 shows the PLEUAR bit assignments.

Table 4-45 PLEUAR bit assignments

Bits	Name	Function
[31:1]	-	RAZ.
[0]	U	User accessibility: 1 = User modes can access PLE registers and execute PLE operations.

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

```
MCR p15, 0, <Rt>, c11, c1, 0; Read PLEAUR
MRC p15, 0, <Rt>, c11, c1, 0; Write PLEAUR
```

4.3.20 Preload Engine Parameters Control Register

The PLEPCR characteristics are:

Purpose Contains PLE control parameters, available only in Privilege modes, to limit the issuing rate and transfer size of the PLE.

Usage constraints The PLEPCR is:

- read/write register
- only accessible in privileged mode
- common to Secure and Non-secure states
- NSACR.PLE controls Non-secure accesses.

Configurations Only available in configurations where the Preload Engine is present, otherwise an Undefined Instruction exception is taken.

Attributes See Table 4-12 on page 4-10.

Figure 4-19 shows the PLEPCR bit assignments.

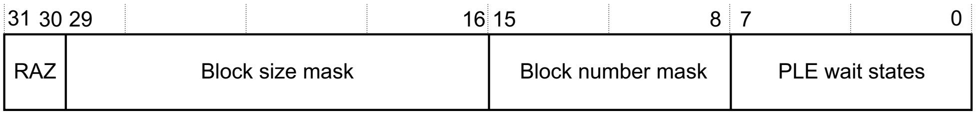


Figure 4-19 PLEPCR bit assignments

Table 4-46 shows the PLEPCR bit assignments.

Table 4-46 PLEPCR bit assignments

Bits	Name	Function
[31:30]	-	RAZ.
[29:16]	Block size mask	Permits Privilege modes to limit the maximum block size for PLE transfers. The transferred block size is: `(Block size) & (Block size mask).` For example, a block size mask of 14’b11111111111111 authorizes the transfer of block sizes with the maximum value of 16k * 4 bytes. A block size mask of 14’b00000000000000 limits block sizes to 1 * 4 bytes.
[15:8]	Block number mask	Permits Privilege modes to limit the maximum number of blocks for a single PLE transfer. The transferred block number is: `(Block number) & (Block number mask).` For example, a block number mask of 8’b11111111 authorizes the transfer of a maximum possible number of 256 blocks. A block number mask of 8’b00000000 limits the transfer to only one block of data.
[7:0]	PLE wait states	Permit Privilege modes to limit the issuing rate of PLD requests performed by the PLE engine to prevent saturation of the external memory bandwidth. PLE wait states specifies the number of cycles inserted between two PLD requests performed by the PLE engine. When PLE wait states is 8’b11111111, the PLE engine can issue one PLD request, a cache line, every 256 cycles. When PLE wait states is 8’b000000000, the PLE engine can issue one PLD request every cycle.

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

```
MCR p15, 0, <Rt>, c11, c1, 1; Read PLEPCR
MRC p15, 0, <Rt>, c11, c1, 1; Write PLEPCR
```

4.3.21 Virtualization Interrupt Register

The VIR characteristics are:

Purpose Indicates that there is a virtual interrupt pending.

Usage constraints The VIR is:
- only accessible in privileged modes
- only accessible in Secure state.

Configurations Available in all configurations.

Attributes See the register summary in Table 4-13 on page 4-10.

The virtual interrupt is delivered as soon as the processor is in NS state. Figure 4-20 shows the VIR bit assignments.

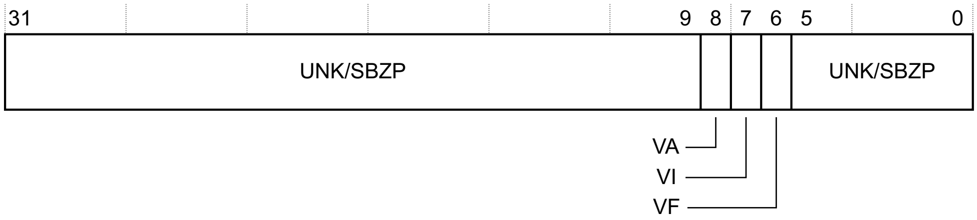


Figure 4-20 VIR bit assignments

Table 4-47 shows the Virtualization Interrupt Register bit assignments.

Table 4-47 Virtualization Interrupt Register bit assignments

Bits	Name	Function
[31:9]	-	UNK/SBZP.
[8]	VA	Virtual Abort bit. When set the corresponding Abort is sent to software in the same way as a normal Abort. The virtual abort happens only when the processor is in Non-secure state.
[7]	VI	Virtual IRQ bit. When set the corresponding IRQ is sent to software in the same way as a normal IRQ. The virtual IRQ happens only when the processor is in Non-secure state.
[6]	VF	Virtual FIQ bit. When set the corresponding FIQ is sent to software in the same way as a normal FIQ. The FIQ happens only when the processor is in Non-secure state.
[5:0]	-	UNK/SBZP.

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

```
MRC p15, 0, <Rd>, c12, c1, 1 ; Read Virtualization Interrupt Register
MCR p15, 0, <Rd>, c12, c1, 1 ; Write Virtualization Interrupt Register
```

4.3.22 Power Control Register

The Power Control Register characteristics are:

Purpose	Enables you to set: • the clock latency for your implementation of the Cortex-A9 processor • dynamic clock gating.
Usage constraints	• a read/write register in Secure state • a read-only register in Non-secure state.
Configurations	Available in all configurations.
Attributes	See the register summary in Table 4-15 on page 4-11.

Figure 4-21 shows the Power Control Register bit assignments.

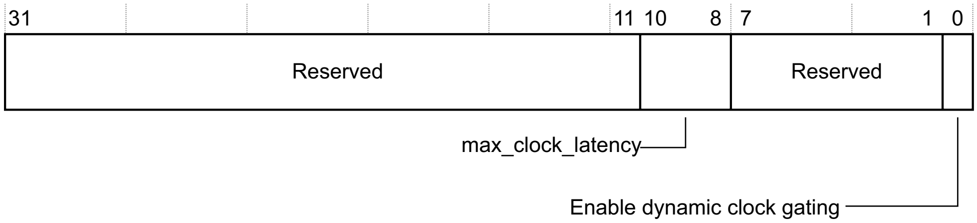


Figure 4-21 Power Control Register bit assignments

Table 4-48 shows the Power Control Register bit assignments.

Table 4-48 Power Control Register bit assignments

Bits	Name	Function
[31:11]	-	Reserved.
[10:8]	max_clk_latency	Samples the value present on the **MAXCLKLATENCY** pins on exit from reset. This value reflects an implementation-specific parameter. ARM strongly recommends that the software does not modify it. The max_clk_latency bits determine the length of the delay between when one of these blocks has its clock cut and the time when it can receive new active signals. If the value determined by max_clk_latency is lower than the real delay, the block that had its clock cut can receive active signals even though it does not have a clock. This can cause the device to malfunction. If the value determined by max_clk_latency is higher than the real delay, the master block waits extra cycles before sending its signals to the block that had its clock cut. This can have some performance impact. When the value is correctly set, the block that had its clock cut receives active signals on the first clock edge of the wake-up. This gives optimum performance.
[7:1]	-	Reserved.
[0]	Enable dynamic clock gating	Disabled at reset.

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

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

4.3.23 NEON Busy Register

The NEON Busy Register characteristics are:

Purpose Enables software to determine if a NEON instruction is executing.

Usage constraints
- a read-only register in Secure state
- a read-only register in Non-secure state.

Configurations Available in all configurations.

Attributes See the register summary in Table 4-15 on page 4-11.

Figure 4-22 shows the NEON Busy Register bit assignments

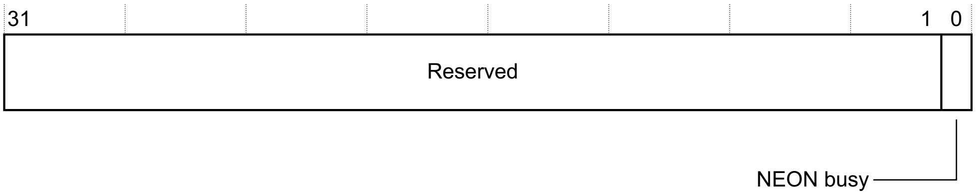


Figure 4-22 NEON Busy Register bit assignments

Table 4-49 shows the NEON Busy Register bit assignments.

Table 4-49 NEON Busy Register bit assignments

Bits	Name	Function
[31:1]	-	Reserved.
[0]	NEON busy	Software can use this to determine if a NEON instruction is executing. This bit is set to 1 if there is a NEON instruction in the NEON pipeline, or in the processor pipeline.

To access the NEON Busy Register, read the CP15 register with:

```
MRC p15,0,<Rd>,c15,c1,0; Read NEON Busy Register
```

4.3.24 Configuration Base Address Register

The Configuration Base Address Register characteristics are:

Purpose Takes the physical base address value at reset.

Usage constraints The Configuration Base Address Register is:
- read/write in secure privileged modes
- read-only in non-secure state
- read-only in User mode.

Configurations In Cortex-A9 uniprocessor implementations the base address is set to zero.

In Cortex-A9 MPCore implementations, the base address is reset to **PERIPHBASE[31:13]** so that software can determine the location of the private memory region.

Attributes See the register summary in Table 4-15 on page 4-11.

Figure 4-23 on page 4-43 shows the Configuration Base Address Register bit assignments.

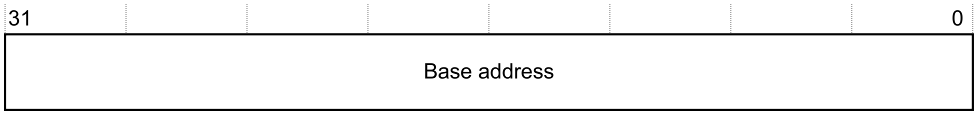


Figure 4-23 Configuration Base Address Register bit assignments

To access the Configuration Base Address Register, read or write the CP15 register with:

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

4.3.25 TLB lockdown operations

TLB lockdown operations enable saving or restoring lockdown entries in the TLB. Table 4-50 shows the defined TLB lockdown operations.

Table 4-50 TLB lockdown operations

Description	Data	Instruction
Select Lockdown TLB Entry for Read	Main TLB Index	`MCR p15,5,<Rd>,c15,c4,2`
Select Lockdown TLB Entry for Write	Main TLB Index	`MCR p15,5,<Rd>,c15,c4,4`
Read Lockdown TLB VA Register	Data	`MRC p15,5,<Rd>,c15,c5,2`
Write Lockdown TLB VA Register	Data	`MCR p15,5,<Rd>,c15,c5,2`
Read Lockdown TLB PA Register	Data	`MRC p15,5,<Rd>,c15,c6,2`
Write Lockdown TLB PA Register	Data	`MCR p15,5,<Rd>,c15,c6,2`
Read Lockdown TLB attributes Register	Data	`MRC p15,5,<Rd>,c15,c7,2`
Write Lockdown TLB attributes Register	Data	`MCR p15,5,<Rd>,c15,c7,2`

The Select Lockdown TLB entry for a read operation is used to select the entry that the data read by a read Lockdown TLB VA/PA/attributes operations are coming from. The Select Lockdown TLB entry for a write operation is used to select the entry that the data write Lockdown TLB VA/PA/attributes data are written to. The TLB PA register must be the last written or read register when accessing TLB lockdown registers. Figure 4-24 shows the bit assignment of the index register used to access the lockdown TLB entries.

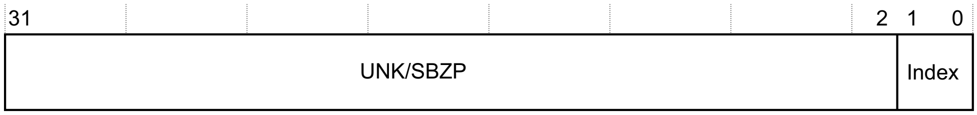


Figure 4-24 Lockdown TLB index bit assignments

Figure 4-25 shows the bit arrangement of the TLB VA Register format.

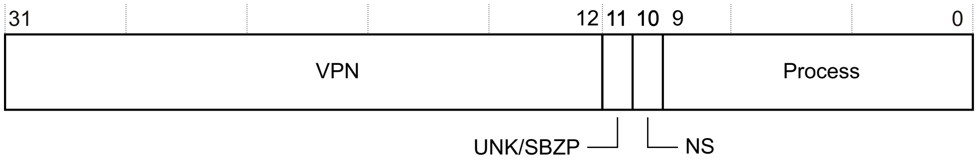


Figure 4-25 TLB VA Register bit assignments

Table 4-51 shows the TLB VA Register bit assignments.

Table 4-51 TLB VA Register bit assignments

Bits	Name	Function
[31:12]	VPN	Virtual page number. Bits of the virtual page number that are not translated as part of the page table translation because the size of the tables is UNPREDICTABLE when read and SBZ when written.
[11]	-	UNK/SBZP.
[10]	NS	NS bit.
[9:0]	Process	Memory space identifier.

Figure 4-26 shows the bit arrangement of the memory space identifier.

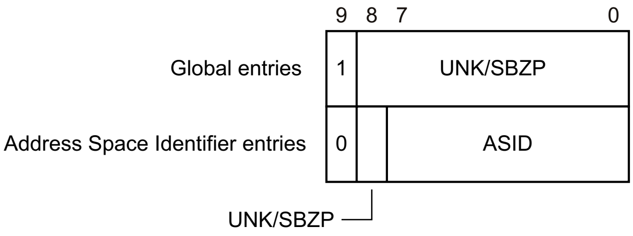


Figure 4-26 Memory space identifier format

Figure 4-27 shows the TLB PA Register bit assignment.

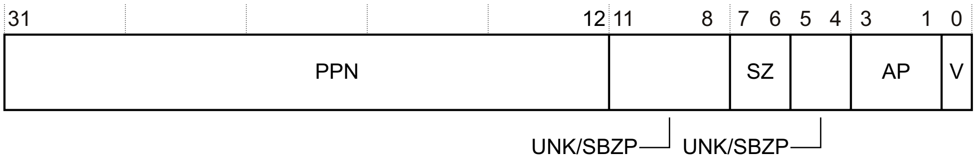


Figure 4-27 TLB PA Register bit assignments

Table 4-52 describes the functions of the TLB PA Register bits.

Table 4-52 TLB PA Register bit assignments

Bits	Name	Function
[31:12]	PPN	Physical Page Number. Bits of the physical page number that are not translated as part of the page table translation are UNPREDICTABLE when read and SBZP when written.
[11:8]	-	UNK/SBZP.
[7:6]	SZ	Region Size: b00 = 16MB Supersection b01 = 4KB page b10 = 64KB page b11 = 1MB section. All other values are reserved.

Table 4-52 TLB PA Register bit assignments (continued)

Bits	Name	Function
[5:4]	-	UNK/SBZP.
[3:1]	AP	Access permission: b000 = All accesses generate a permission fault b001 = Supervisor access only, User access generates a fault b010 = Supervisor read/write access, User write access generates a fault b011 = Full access, no fault generated b100 = Reserved b101 = Supervisor read-only b110 = Supervisor/User read-only b111 = Supervisor/User read-only.
[0]	V	Value bit. Indicates that this entry is locked and valid.

Figure 4-28 shows the bit assignments of the TLB Attributes Register.

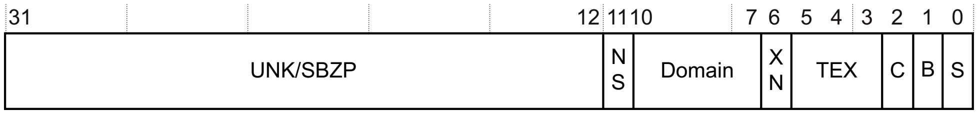


Figure 4-28 Main TLB Attributes Register bit assignments

Table 4-53 shows the TLB Attributes Register bit assignments. The Cortex-A9 processor does not support subpages.

Table 4-53 TLB Attributes Register bit assignments

Bits	Name	Function
[31:12]	-	UNK/SBZP.
[11]	NS	Non-secure description.
[10:7]	Domain	Domain number of the TLB entry.
[6]	XN	Execute Never attribute.
[5:3]	TEX	Region type encoding. See the *ARM Architecture Reference Manual*.
[2:1]	CB	
[0]	S	Shared attribute.

Invalidate TLB Entries on ASID Match

This is a single interruptible operation that invalidates all TLB entries that match the provided *Address Space Identifier* (ASID) value. This function invalidates locked entries. Entries marked as global are not invalidated by this function.

In the Cortex-A9 processor, this operation takes several cycles to complete and the instruction is interruptible. When interrupted the r14 state is set to indicate that the `MCR` instruction has not executed. Therefore, r14 points to the address of the `MCR` + 4. The interrupt routine then automatically restarts at the `MCR` instruction. If this operation is interrupted and later restarted, any entries fetched into the TLB by the interrupt that uses the provided ASID are invalidated by the restarted invalidation.

Chapter 5
Jazelle DBX registers

This chapter introduces the CP14 coprocessor and describes the non-debug use of CP14. It contains the following sections:

- *About coprocessor CP14* on page 5-2
- *CP14 Jazelle register summary* on page 5-3
- *CP14 Jazelle register descriptions* on page 5-4.

5.1 About coprocessor CP14

The non-debug use of coprocessor CP14 provides support for the hardware acceleration of Java bytecodes.

See the *ARM Architecture Reference Manual* for more information.

5.2 CP14 Jazelle register summary

In the Cortex-A9 implementation of the Jazelle Extension:

- Jazelle state is supported.
- The `BXJ` instruction enters Jazelle state.

Table 5-1 shows the CP14 Jazelle registers. For all Jazelle register accesses, CRm and Op2 are zero. All Jazelle registers are 32 bits wide.

Table 5-1 CP14 Jazelle registers summary

Op1	CRn	Name	Type	Reset	Page
7	0	Jazelle ID Register (JIDR)	RW[a]	`0xF4100168`	page 5-4
7	1	Jazelle OS Control Register (JOSCR)	RW	-	page 5-5
7	2	Jazelle Main Configuration Register (JMCR)	RW	-	page 5-6
7	3	Jazelle Parameters Register	RW	-	page 5-7
7	4	Jazelle Configurable Opcode Translation Table Register	WO	-	page 5-8

[a] See *Write operation of the JIDR* on page 5-5 for the effect of a write operation.

See the *ARM Architecture Reference Manual* for information about the Jazelle Extension.

5.3 CP14 Jazelle register descriptions

The following sections describe the CP14 Jazelle DBX registers arranged in numerical order, as shown in Table 5-1 on page 5-3:

- *Jazelle ID Register*
- *Jazelle Operating System Control Register* on page 5-5
- *Jazelle Main Configuration Register* on page 5-6
- *Jazelle Parameters Register* on page 5-7
- *Jazelle Configurable Opcode Translation Table Register* on page 5-8.

5.3.1 Jazelle ID Register

The JIDR characteristics are:

Purpose Enables software to determine the implementation of the Jazelle Extension provided by the processor.

Usage constraints The JIDR is:

- accessible in privileged modes.
- also accessible in User mode if the CD bit is clear. See *Jazelle Operating System Control Register* on page 5-5.

Configurations Available in all configurations.

Attributes See the register summary in Table 5-1 on page 5-3.

Figure 5-1 shows the JIDR bit assignments.

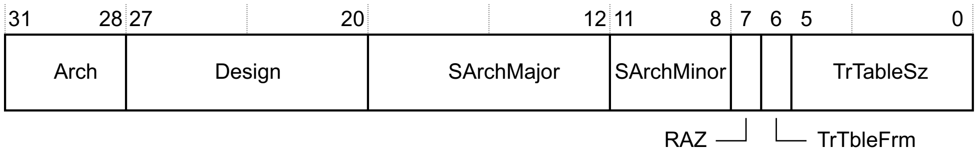


Figure 5-1 JIDR bit assignments

Table 5-2 shows the JIDR bit assignments.

Table 5-2 JIDR bit assignments

Bits	Name	Function
[31:28]	Arch	This uses the same architecture code that appears in the Main ID register.
[27:20]	Design	Contains the implementer code of the designer of the subarchitecture.
[19:12]	SArchMajor	The subarchitecture code.
[11:8]	SArchMinor	The subarchitecture minor code.
[7]	-	RAZ.
[6]	TrTbleFrm	Indicates the format of the Jazelle Configurable Opcode Translation Table Register.
[5:0]	TrTbleSz	Indicates the size of the Jazelle Configurable Opcode Translation Table Register.

To access the JIDR, read the CP14 register with:

```
MRC p14, 7, <Rd>, c0, c0, 0; Read Jazelle Identity Register
```

Write operation of the JIDR

A write to the JIDR clears the translation table. This has the effect of making all configurable opcodes executed in software only. See *Jazelle Configurable Opcode Translation Table Register* on page 5-8.

5.3.2 Jazelle Operating System Control Register

The JOSCR characteristics are:

Purpose	Enables operating systems to control access to Jazelle Extension hardware.
Usage constraints	The JOSCR is: • only accessible in privileged modes. • set to zero after a reset and must be written in privileged modes.
Configurations	Available in all configurations.
Attributes	See the register summary in Table 5-1 on page 5-3.

Figure 5-2 shows the JOSCR bit assignments.

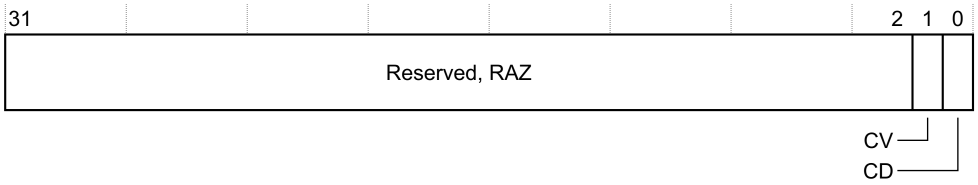


Figure 5-2 JOSCR bit assignments

Table 5-3 shows the JOSCR bit assignments.

Table 5-3 JOSCR bit assignments

Bits	Name	Function
[31:2]	-	Reserved, RAZ.
[1]	CV	Configuration Valid bit. 0 = The Jazelle configuration is invalid. Any attempt to enter Jazelle state when the Jazelle hardware is enabled: • generates a configuration invalid Jazelle exception • sets this bit, marking the Jazelle configuration as valid. 1 = The Jazelle configuration is valid. Entering Jazelle state succeeds when the Jazelle hardware is enabled. The CV bit is automatically cleared on an exception.
[0]	CD	Configuration Disabled bit. 0 = Jazelle configuration in User mode is enabled: • reading the JIDR succeeds • reading any other Jazelle configuration register generates an Undefined Instruction exception • writing the JOSCR generates an Undefined Instruction exception • writing any other Jazelle configuration register succeeds. 1 = Jazelle configuration from User mode is disabled: • reading any Jazelle configuration register generates an Undefined Instruction exception • writing any Jazelle configuration register generates an Undefined Instruction exception.

To access the JOSCR, read or write the CP14 register with:

```
MRC p14, 7, <Rd>, c1, c0, 0; Read JOSCR
MCR p14, 7, <Rd>, c1, c0, 0; Write JOSCR
```

5.3.3 Jazelle Main Configuration Register

The JMCR characteristics are:

Purpose	Describes the Jazelle hardware configuration and its behavior.
Usage constraints	Only accessible in privileged modes.
Configurations	Available in all configurations.
Attributes	See the register summary in Table 5-1 on page 5-3.

Figure 5-3 shows the JMCR bit assignments.

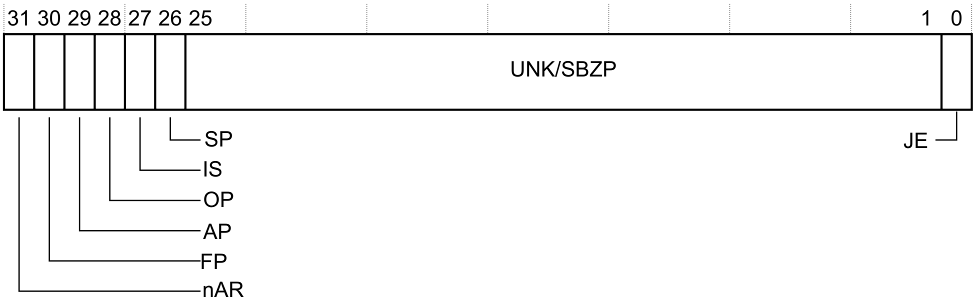


Figure 5-3 JMCR bit assignments

Table 5-4 shows the JMCR bit assignments.

Table 5-4 JMCR bit assignments

Bits	Name	Function
[31]	nAR	*not Array Operations* (nAR) bit. 0 = Execute array operations in hardware, if implemented. Otherwise, call the appropriate handlers in the VM Implementation Table. 1 = Execute all array operations by calling the appropriate handlers in the VM Implementation Table.
[30]	FP	The FP bit controls how the Jazelle hardware executes JVM floating-point opcodes: 0 = Execute all JVM floating-point opcodes by calling the appropriate handlers in the VM Implementation Table. 1 = Execute JVM floating-point opcodes by issuing `VFP` instructions, where possible. Otherwise, call the appropriate handlers in the VM Implementation Table. In this implementation FP is set to zero and is read-only.
[29]	AP	The *Array Pointer* (AP) bit controls how the Jazelle hardware treats array references on the operand stack: 0 = Array references are treated as handles. 1 = Array references are treated as pointers.
[28]	OP	The *Object Pointer* (OP) bit controls how the Jazelle hardware treats object references on the operand stack: 0 = Object references are treated as handles. 1 = Object references are treated as pointers.

Table 5-4 JMCR bit assignments (continued)

Bits	Name	Function
[27]	IS	The *Index Size* (IS) bit specifies the size of the index associated with quick object field accesses: 0 = Quick object field indices are 8 bits. 1 = Quick object field indices are 16 bits.
[26]	SP	The *Static Pointer* (SP) bit controls how the Jazelle hardware treats static references: 0 = Static references are treated as handles. 1 = Static references are treated as pointers.
[25:1]	-	UNK/SBZP.
[0]	JE	The *Jazelle Enable* (JE) bit controls whether the Jazelle hardware is enabled, or is disabled: 0 = The Jazelle hardware is disabled: • `BXJ` instructions behave like `BX` instructions • setting the J bit in the CPSR generates a Jazelle-Disabled Jazelle exception. 1 = The Jazelle hardware is enabled: • `BXJ` instructions enter Jazelle state • setting the J bit in the CPSR enters Jazelle state.

To access the JMCR, read or write the CP14 register with:

```
MRC p14, 7, <Rd>, c2, c0, 0; Read JMCR
MCR p14, 7, <Rd>, c2, c0, 0; Write JMCR
```

5.3.4 Jazelle Parameters Register

The Jazelle Parameters Register characteristics are:

Purpose	Describes the configuration parameters of the Jazelle hardware.
Usage constraints	Only accessible in privileged modes.
Configurations	Available in all configurations.
Attributes	See the register summary in Table 5-1 on page 5-3.

Figure 5-4 shows the Jazelle Parameters Register bit assignments.

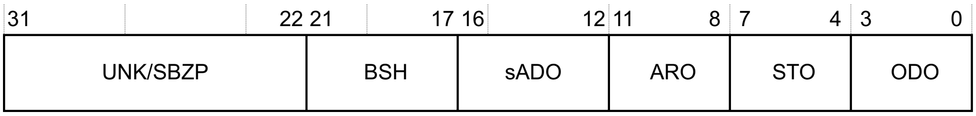


Figure 5-4 Jazelle Parameters Register bit assignments

Table 5-5 shows the Jazelle Parameters Register bit assignments.

Table 5-5 Jazelle Parameters Register bit assignments

Bits	Name	Function
[31:22]	-	UNK/SBZP.
[21:17]	BSH	The *Bounds SHift* (BSH) bits contain the offset, in bits, of the array bounds (number of items in the array) within the array descriptor word.
[16:12]	sADO	The *signed Array Descriptor Offset* (sADO) bits contain the offset, in words, of the array descriptor word from an array reference. The offset is a sign-magnitude signed quantity: • Bit [16] gives the sign of the offset. The offset is positive if the bit is clear, and negative if the bit is set. • Bits [15:12] give the absolute magnitude of the offset.
[11:8]	ARO	The *Array Reference Offset* (ARO) bits contain the offset, in words, of the array data or the array data pointer from an array reference.
[7:4]	STO	The *STatic Offset* (STO) bits contain the offset, in words, of the static or static pointer from a static reference.
[3:0]	ODO	The *Object Descriptor Offset* (ODO) bits contain the offset, in words, of the field from the base of an object data block.

To access the Jazelle Parameters Register, read or write the CP14 register with:

```
MRC p14, 7, <Rd>, c3, c0, 0; Read Jazelle Parameters Register
MCR p14, 7, <Rd>, c3, c0, 0; Write Jazelle Parameters Register
```

5.3.5 Jazelle Configurable Opcode Translation Table Register

The Jazelle Configurable Opcode Translation Table Register characteristics are:

Purpose	Provides translations between the configurable opcodes in the range `0xCB`-`0xFD` and the operations that are provided by the Jazelle hardware.
Usage constraints	Only accessible in privileged modes.
Configurations	Available in all configurations.
Attributes	See the register summary in Table 5-1 on page 5-3.

Figure 5-5 shows the Jazelle Configurable Opcode Translation Table Register bit assignments.

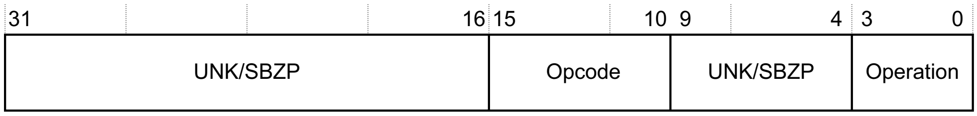


Figure 5-5 Jazelle Configurable Opcode Translation Table Register bit assignments

Table 5-6 shows the Jazelle Configurable Opcode Translation Table Register bit assignments.

Table 5-6 Jazelle Configurable Opcode Translation Table Register bit assignments

Bits	Name	Function
[31:16]	-	UNK/SBZP
[15:10]	Opcode	Contains the bottom bits of the configurable opcode
[9:4]	-	UNK/SBZP
[3:0]	Operation	Contains the code for the operation `0x0-0x9`

To access the Jazelle Configurable Opcode Translation Table Register, write the CP14 register with:

```
MCR p14, 7, <Rd>, c4, c0, 0;  Write Jazelle Configurable Opcode Translation
                              Table Register
```

Chapter 6
Memory Management Unit

This chapter describes the MMU. It contains the following sections:

- *About the MMU* on page 6-2
- *TLB Organization* on page 6-4
- *Memory access sequence* on page 6-6
- *MMU enabling or disabling* on page 6-7
- *External aborts* on page 6-8.

6.1 About the MMU

The MMU works with the L1 and L2 memory system to translate virtual addresses to physical addresses. It also controls accesses to and from external memory.

The *Virtual Memory System Architecture version 7* (VMSAv7) features include the following:

- page table entries that support 4KB, 64KB, 1MB, and 16MB
- 16 domains
- global and address space identifiers to remove the requirement for context switch TLB flushes
- extended permissions check capability.

See the *ARM Architecture Reference Manual* for a full architectural description of the VMSAv7.

The processor implements the ARMv7-A MMU enhanced with Security Extensions and multiprocessor extensions to provide address translation and access permission checks. The MMU controls table walk hardware that accesses translation tables in main memory. The MMU enables fine-grained memory system control through a set of virtual-to-physical address mappings and memory attributes.

Note

In VMSAv7 first level descriptor formats page table base address bit [9] is implementation-defined. In Cortex-A9 processor designs this bit is unused.

The MMU features include the following:

- Instruction side micro TLB
    - 32 fully associative entries.
- Data side micro TLB
    - 32 fully associative entries.
- Unified main TLB
    - unified, 2-way associative, 2x32 entry TLB for the 64-entry TLB and 2x64 entry TLB for the 128-entry TLB.
    - 4 lockable entries using the lock-by-entry model.
    - supports hardware page table walks to perform lookups in the L1 data cache.

6.1.1 Memory Management Unit

The MMU performs the following operations:

- checking of Virtual Address and ASID
- checking of domain access permissions
- checking of memory attributes
- virtual-to-physical address translation
- support for four page (region) sizes
- mapping of accesses to cache, or external memory
- TLB loading for hardware and software.

Domains

The Cortex-A9 processor supports 16 access domains.

TLB

The Cortex-A9 processor implements a 2-level TLB structure. Four entries in the main TLB are lockable.

ASIDs

Main TLB entries can be global, or can be associated with particular processes or applications using *Address Space Identifiers* (ASIDs). ASIDs enable TLB entries to remain resident during context switches, avoiding the requirement of reloading them subsequently. See *Invalidate TLB Entries on ASID Match* on page 4-45.

System control coprocessor

TLB maintenance and configuration operations are controlled through a dedicated coprocessor, CP15, integrated within the processor. This coprocessor provides a standard mechanism for configuring the level one memory system.

6.2 TLB Organization

The following sections describe the organization of the TLB:

- *Micro TLB*
- *Main TLB*.

6.2.1 Micro TLB

The first level of caching for the page table information is a micro TLB of 32 entries that is implemented on each of the instruction and data sides. These blocks provide a fully associative lookup of the virtual addresses in a single **CLK** signal cycle.

The micro TLB returns the physical address to the cache for the address comparison, and also checks the protection attributes to signal either a Prefetch Abort or a Data Abort.

All main TLB related operations affect both the instruction and data micro TLBs, causing them to be flushed. In the same way, any change of the Context ID Register causes the micro TLBs to be flushed.

6.2.2 Main TLB

The main TLB catches the misses from the micro TLBs. It also provides a centralized source for lockable translation entries.

Accesses to the main TLB take a variable number of cycles, according to competing requests from each of the micro TLBs and other implementation-dependent factors. Entries in the lockable region of the main TLB are lockable at the granularity of a single entry. As long as the lockable region does not contain any locked entries, it can be allocated with non-locked entries to increase overall main TLB storage size.

The main TLB is implemented as a combination of:

- a fully-associative, lockable array of four elements
- a 2-way associative structure of 2x32 or 2x64 entries.

TLB match process

Each TLB entry contains a virtual address, a page size, a physical address, and a set of memory properties. Each is marked as being associated with a particular application space, or as global for all application spaces. CONTEXIDR determines the selected application space. A TLB entry matches if bits [31:N] of the modified virtual address match, where N is $\log_2$ of the page size for the TLB entry. It is either marked as global, or the ASID matched the current ASID.

A TLB entry matches when these conditions are true:

- its virtual address matches that of the requested address
- its Non-secure TLB ID (NSTID) matches the Secure or Non-secure state of the MMU request
- its ASID matches the current ASID or is global.

The operating system must ensure that, at most, one TLB entry matches at any time.

Supersections, sections, and large pages are supported to permit mapping of a large region of memory while using only a single entry in a TLB. If no mapping for an address is found in the TLB, then the translation table is automatically read by hardware and a mapping is placed in the TLB.

TLB lockdown

The TLB supports the TLB lock-by-entry model as described in the *ARM Architecture Reference Manual*. See *TLB lockdown operations* on page 4-43 for more information.

6.3 Memory access sequence

When the processor generates a memory access, the MMU:

1. Performs a lookup for the requested virtual address and current ASID and security state in the relevant instruction or data micro TLB.
2. If there is a miss in the micro TLB, performs a lookup for the requested virtual address and current ASID and security state in the main TLB.
3. If there is a miss in the main TLB, performs a hardware translation table walk.

You can configure the MMU to perform hardware translation table walks in cacheable regions by setting the IRGN bits in the Translation Table Base Registers. If the encoding of the IRGN bits is write-back, then an L1 data cache lookup is performed and data is read from the data cache. If the encoding of the IRGN bits is write-through or non-cacheable then an access to external memory is performed.

The MMU might not find a global mapping, or a mapping for the selected ASID, with a matching *Non-secure TLB ID* (NSTID) for the virtual address in the TLB. In this case, the hardware does a translation table walk if the translation table walk is enabled by the PD0 or PD1 bit in the TTB Control Register. If translation table walks are disabled, the processor returns a Section Translation fault.

If the MMU finds a matching TLB entry, it uses the information in the entry as follows:

1. The access permission bits and the domain determine if the access is enabled. If the matching entry does not pass the permission checks, the MMU signals a memory abort. See the *ARM Architecture Reference Manual* for a description of access permission bits, abort types and priorities, and for a description of the IFSR and *Data Fault Status Register* (DFSR).
2. The memory region attributes specified in both the TLB entry and the CP15 c10 remap registers control the cache and write buffer, and determine if the access is
    - Secure or Non-secure
    - Shared or not
    - Normal memory, Device, or Strongly-ordered.
3. The MMU translates the virtual address to a physical address for the memory access.

If the MMU does not find a matching entry, a hardware table walk occurs.

6.4 MMU enabling or disabling

You can enable or disable the MMU as described in the *ARM Architecture Reference Manual*.

6.5 External aborts

External memory errors are defined as those that occur in the memory system rather than those that are detected by the MMU. External memory errors are expected to be extremely rare. External aborts are caused by errors flagged by the AXI interfaces when the request goes external to the processor. External aborts can be configured to trap to Monitor mode by setting the EA bit in the Secure Configuration Register.

6.5.1 External aborts on data read or write

Externally generated errors during a data read or write can be asynchronous. This means that the r14_abt on entry into the abort handler on such an abort might not hold the address of the instruction that caused the exception.

The DFAR is UNPREDICTABLE when an asynchronous abort occurs.

In the case of a load multiple or store multiple operation, the address captured in the DFAR is that of the address that generated the synchronous external abort.

6.5.2 Synchronous and asynchronous aborts

To determine a fault type, read the DFSR for a data abort or the IFSR for an instruction abort.

The processor supports an Auxiliary Fault Status Register for software compatibility reasons only. The processor does not modify this register because of any generated abort.

Chapter 7
Level 1 Memory System

This chapter describes the L1 Memory System. It contains the following sections:

7.1 About the L1 memory system

The L1 memory system has:

- separate instruction and data caches each with a fixed line length of 32 bytes
- 64-bit data paths throughout the memory system
- support for four sizes of memory page
- export of memory attributes for external memory systems
- support for Security Extensions.

The data side of the L1 memory system has:

- two 32-byte linefill buffers and one 32-byte eviction buffer
- a 4-entry, 64-bit merging store buffer.

Note

You must invalidate the instruction cache, the data cache, TLB, and BTAC before using them.

7.1.1 Memory system

This section describes:

- *Cache features*
- *Instruction cache features*
- *Data cache features* on page 7-3
- *Store buffer* on page 7-3.

Cache features

The Cortex-A9 processor has separate instruction and data caches. The caches have the following features:

- Each cache can be disabled independently. See *System Control Register* on page 4-24.
- Both caches are 4-way set-associative.
- The cache line length is eight words.
- On a cache miss, critical word first filling of the cache is performed.
- You can configure the instruction and data caches independently during implementation to sizes of 16KB, 32KB, or 64KB.
- To reduce power consumption, the number of full cache reads is reduced by taking advantage of the sequential nature of many cache operations. If a cache read is sequential to the previous cache read, and the read is within the same cache line, only the data RAM set that was previously read is accessed.

Instruction cache features

The instruction cache has the following features:

- The instruction cache is virtually indexed and physically tagged.
- Instruction cache replacement policy is either pseudo round-robin or pseudo random.

Data cache features

The data cache has the following features:

- The data cache is physically indexed and physically tagged.
- Data cache replacement policy is pseudo random.
- Both data cache read misses and write misses are non-blocking with up to four outstanding data cache read misses and up to four outstanding data cache write misses being supported.

Store buffer

The Cortex-A9 processor has a store buffer with four 64-bit slots with data merging capability.

7.2 Security Extensions support

The Cortex-A9 processor supports the Security Extensions, and exports the Secure or Non-secure status of its memory requests to the memory system. See the *ARM Architecture Reference Manual* for more information.

7.3 About the L1 instruction side memory system

The L1 instruction side memory system provides an instruction stream to the Cortex-A9 processor. To increase overall performance and to reduce power consumption, it contains the following functionality:

- dynamic branch prediction
- instruction caching.

Figure 7-1 shows this.

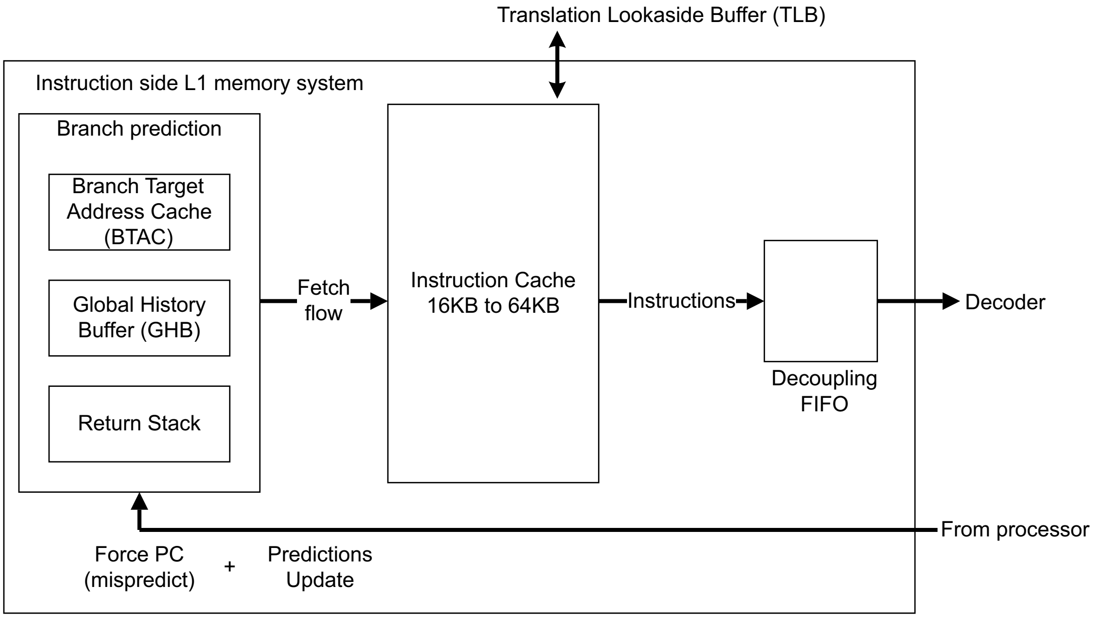


Figure 7-1 Branch prediction and instruction cache

The ISide comprises the following:

The Prefetch Unit (PFU)

The Prefetch Unit implements a 2-level prediction mechanism, comprising:

- a 2-way BTAC of 512 entries organized as 2-way x 256 entries implemented in RAMs
- a *Global History Buffer* (GHB) containing 4096 2-bit predictors implemented in RAMs
- a return stack with eight 32-bit entries.

  The prediction scheme is available in ARM state, Thumb state, ThumbEE state, and Jazelle state. It is also capable of predicting state changes from ARM to Thumb, and from Thumb to ARM. It does not predict
  - any other state changes
  - any instruction that changes the mode of the processor.

  See *Program flow prediction* on page 7-6.

Instruction Cache Controller

The instruction cache controller fetches the instructions from memory depending on the program flow predicted by the prefetch unit.

The instruction cache is 4-way set associative. It comprises the following features:

- configurable sizes of 16KB, 32KB, or 64KB
- *Virtually Indexed Physically Tagged* (VIPT)
- 64-bit native accesses to provide up to four instructions per cycle to the prefetch unit
- Security Extensions support
- no lockdown support.

7.3.1 Enabling program flow prediction

You can enable program flow prediction by setting the Z bit in the CP15 c1 Control Register to 1. See *System Control Register* on page 4-24. Before switching program flow prediction on, you must perform a BTAC flush operation.

This has the additional effect of setting the GHB into a known state.

7.3.2 Program flow prediction

The following sections describe program flow prediction:

- *Predicted and nonpredicted instructions*
- *Thumb state conditional branches*
- *Return stack predictions* on page 7-7.

Predicted and nonpredicted instructions

This section shows the instructions that the processor predicts. Unless otherwise specified, the list applies to ARM, Thumb, ThumbEE, and Jazelle instructions.As a general rule, the flow prediction hardware predicts all branch instructions regardless of the addressing mode, including:

- conditional branches
- unconditional branches
- indirect branches
- PC-destination data-processing operations
- branches that switch between ARM and Thumb states.

However, some branch instructions are nonpredicted:

- Branches that switch between states (except ARM to Thumb transitions, and Thumb to ARM transitions).
- Instructions with the S suffix are not predicted, because they are typically used to return from exceptions and have side effects that can change privilege mode and security state.
- All mode changing instructions.

Thumb state conditional branches

In Thumb state, a branch that is normally encoded as unconditional can be made conditional by inclusion in an *If-Then-Else* (ITE) block. Then it is treated as a normal conditional branch.

Return stack predictions

The return stack stores the address and the instruction execution state of the instruction after a function-call type branch instruction. This address is equal to the link register value stored in r14. The following instructions cause a return stack push if predicted:

- `BL` immediate
- `BLX(1)` immediate
- `BLX(2)` register
- `HBL` (ThumbEE state)
- `HBLP` (ThumbEE state).

The following instructions cause a return stack pop if predicted:

- `BX r14`
- `MOV pc, r14`
- `LDM r13, {…pc}`
- `LDR pc, [r13]`.

The `LDR` instruction can use any of the addressing modes, as long as r13 is the base register. Additionally, in ThumbEE state you can also use r9 as a stack pointer so the `LDR` and `LDM` instructions with pc as a destination and r9 as a base register are also treated as a return stack pop.

Because return-from-exception instructions can change processor privilege mode and security state, they are not predicted. This includes the `LDM(3)` instruction, and the `MOVS pc, r14` instruction.

7.4 About the L1 data side memory system

The L1 data cache is organized as a physically indexed and physically tagged cache. The micro TLB produces the physical address from the virtual address before performing the cache access.

7.4.1 Local Monitor

The L1 memory system of the Cortex-A9 processor has a local monitor. This is a 2-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 to construct semaphores, ensuring synchronization between different processes running on the processor, and also between different processors that are using the same coherent memory locations for the semaphore.

——— **Note** ———

A store exclusive can generate an MMU fault or cause the processor to take a data watchpoint exception regardless of the state of the local monitor. See Table 10-8 on page 10-11

———————

See the *ARM Architecture Reference Manual* for more information about these instructions.

Treatment of intervening STR operations

In cases where there is an intervening `STR` operation in an `LDREX`/`STREX` code sequence, the intermediate `STR` does not produce any effect on the internal exclusive monitor. The local monitor is in the Exclusive Access state after the `LDREX`, remains in the Exclusive Access state after the `STR`, and returns to the Open Access state only after the `STREX`.

LDREX/STREX operations using different sizes

In cases where the LDREX and STREX operations are of different sizes a check is performed to ensure that the tagged address bytes match or are within the size range of the store operation.

The granularity of the tagged address for an `LDREX` instruction is eight words, aligned on an 8-word boundary. This size is implementation-defined, and as such, software must not rely on this granularity remaining constant on other ARM cores.

7.4.2 External aborts handling

The L1 data cache handles two types of external abort depending on the attributes of the memory region of the access:

- All Strongly-ordered accesses use the synchronous abort mechanism.
- All Cacheable, Device, and Normal Non-cacheable memory requests use the asynchronous abort mechanism. For example, an abort returned on a read miss, issuing a linefill, is flagged as asynchronous.

7.4.3 Cortex-A9 behavior for Normal Memory Cacheable memory regions

Depending on its configuration settings, and on the inner attributes specified in the page table descriptors, the Cortex-A9 cacheable accesses behave as follows:

SCTLR.C=0 The Cortex-A9 L1 Data Cache is not enabled. All memory accesses to Normal Memory Cacheable regions are treated as Normal Memory Non-Cacheable, without lookup and without allocation in the L1 Data Cache.

SCTLR.C=1 The Cortex-A9 Data Cache is enabled. Some Cacheable accesses are still treated as Non-Cacheable:

- all pages marked as Write-Through are treated as Non-Cacheable
- if ACTLR.SMP=0, all pages marked as Shared are treated as Non-Cacheable.

Note

ARUSER[4:0] and **AWUSER[4:0]** directly reflect the value of the Inner attributes and Shared attribute as defined in the corresponding page descriptor. They do not reflect how the Cortex-A9 processor interprets them, and whether the access was treated as Cacheable or not.

7.5 About DSB

The Cortex-A9 processor only implements the SY option of the `DSB` instruction. All other DSB options execute as a full system DSB operation, but software must not rely on this operation.

7.6 Data prefetching

This section describes:

- *The PLD instruction*
- *Data prefetching and monitoring*.

7.6.1 The PLD instruction

All `PLD` instructions are handled in a dedicated unit in the Cortex-A9 processor with dedicated resources. This avoids using resources in the integer core or the Load Store Unit.

7.6.2 Data prefetching and monitoring

The Cortex-A9 data cache implements an automatic prefetcher that monitors cache misses done by the processor. This unit can monitor and prefetch two independent data streams. It can be activated in software using a CP15 Auxiliary Control Register bit. See *Auxiliary Control Register* on page 4-27.

When the software issues a `PLD` instruction the PLD prefetch unit always takes precedence over requests from the data prefetch mechanism. Prefetched lines in the speculative prefetcher can be dropped before they are allocated. `PLD` instructions are always executed and never dropped.

7.7 Parity error support

If your configuration implements parity error support, the features are as follows:

- the parity scheme is even parity. For byte 0000000 parity is 0.
- each RAM in the design generates parity information. As a general rule each RAM byte generates one parity bit. Where RAM bit width is not a multiple of eight, the remaining bits produce one parity bit.

  There is also support for parity bit-writable data.
- RAM arrays in a design with parity support store parity information alongside the data in the RAM banks. As a result RAM arrays are wider when your design implements parity support.
- The Cortex-A9 logic includes the additional parity generation logic and the parity checking logic.

Figure 7-2 shows the parity support design features and stages. In stages 1 and 2 RAM writes and parity generation take place in parallel. RAM reads and parity checking take place in parallel in stages 3 and 4.

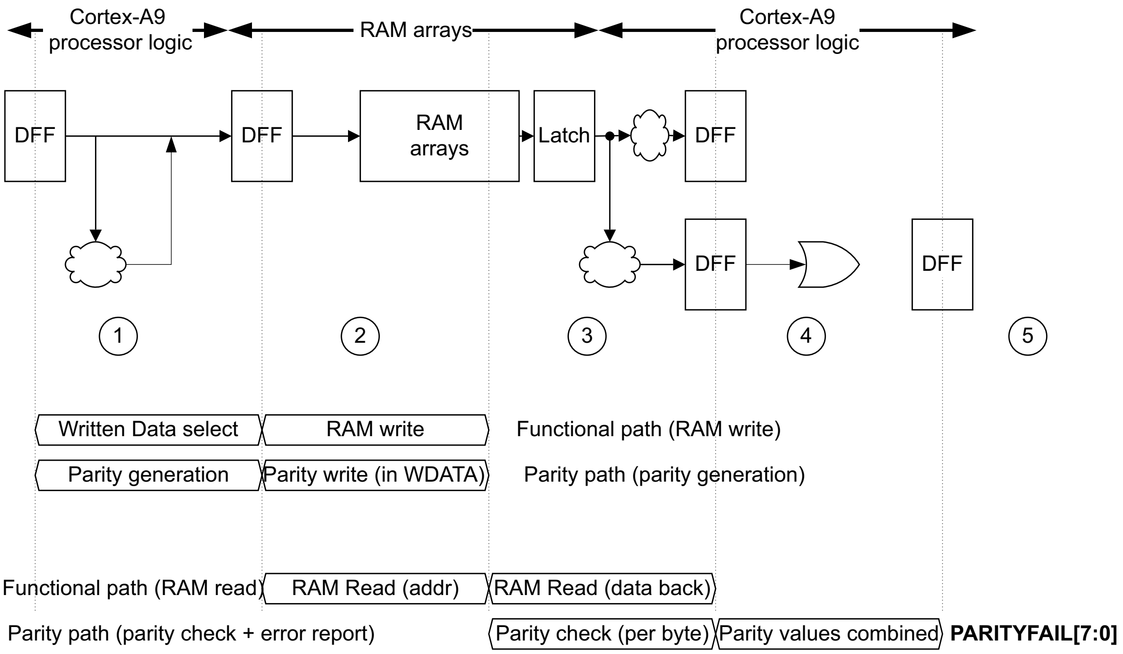


Figure 7-2 Parity support

The output signals **PARITYFAIL[7:0]** report parity errors. Typically, **PARITYFAIL[7:0]** reports parity errors three clock cycles after the corresponding RAM read.

Note

This is not a precise error detection scheme. Designers can implement a precise error detection scheme by adding address register pipelines for RAMs. It is the responsibility of the designer to correctly implement this logic.

7.7.1 GHB and BTAC data corruption

The scheme provides parity error support for GHB RAMs and BTAC RAMs but this support has limited diagnostic value. Corruption in GHB data or BTAC data does not generate functional errors in the Cortex-A9 processor. Corruption in GHB data or BTAC data results in a branch misprediction, that is detected and corrected.

Chapter 8

Level 2 Memory Interface

This chapter describes the L2 memory interface. It contains the following sections:

8.1 About the Cortex-A9 L2 interface

This section describes the Cortex-A9 L2 interface in:

- *Cortex-A9 L2 interface*
- *Supported AXI transfers* on page 8-3
- *AXI transaction IDs* on page 8-3
- *AXI USER bits* on page 8-4.
- *Exclusive L2 cache* on page 8-5.

8.1.1 Cortex-A9 L2 interface

The Cortex-A9 L2 interface consists of two 64-bit wide AXI bus masters:

- M0 is the data side bus
- M1 is the instruction side bus and has no write channels.

Table 8-1 shows the AXI master 0 interface attributes.

Table 8-1 AXI master 0 interface attributes

Attribute	Format
Write issuing capability	12, including: • eight noncacheable writes • four evictions.
Read issuing capability	10, including: • six linefill reads. • four noncacheable read
Combined issuing capability	22
Write ID capability	2
Write interleave capability	1
Write ID width	2
Read ID capability	3
Read ID width	2

Table 8-2 shows the AXI master 1 interface attributes.

Table 8-2 AXI master 1 interface attributes

Attribute	Format
Write issuing capability	None
Read issuing capability	4 instruction reads
Combined issuing capability	4
Write ID capability	None
Write interleave capability	None

Table 8-2 AXI master 1 interface attributes (continued)

Attribute	Format
Write ID width	None
Read ID capability	4
Read ID width	2

Note

The numbers in Table 8-1 on page 8-2 and Table 8-2 on page 8-2 are the theoretical maximums for the Cortex-A9 MPCore processor. A typical system is unlikely to reach these numbers. ARM recommends that you perform profiling to tailor your system resources appropriately for optimum performance.

The AXI protocol and meaning of each AXI signal are not described in this document. For more information see *AMBA AXI Protocol v1.0 Specification*.

8.1.2 Supported AXI transfers

The Cortex-A9 master ports generate only a subset of all possible AXI transactions.

For cacheable transactions:

- WRAP4 64-bit for read transfers (linefills)
- INCR4 64-bit for write transfers (evictions)

For noncacheable transactions:

- INCR N (N:1- 9) 64-bit read transfers
- INCR 1 for 64-bit write transfers
- INCR N (N:1-16) 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, 64-bit exclusive read/write transfers
- INCR 1 for 8-bit and 32-bit read/write (locked) for swap

The following points apply to AXI transactions:

- WRAP bursts are only read transfers, 64-bit, 4 transfers
- INCR 1 can be any size for read or write
- INCR burst (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.

8.1.3 AXI transaction IDs

The AXI ID signal is encoded as follows:

- For the data side read bus, **ARIDM0**, is encoded as follows:
  - 2'b00 for noncacheable accesses
  - 2'b01 is unused
  - 2'b10 for linefill 0 accesses
  - 2'b11 for linefill 1 accesses.

- For the instruction side read bus, **ARIDM1**, is encoded as follows:
  - 2'b00 for outstanding transactions
  - 2'b01 for outstanding transactions
  - 2'b10 for outstanding transactions
  - 2'b11 for outstanding transactions.
- For the data side write bus, **AWIDM0**, is encoded as follows:
  - 2'b00 for noncacheable accesses
  - 2'b01 is unused
  - 2'b10 for linefill 0 evictions
  - 2'b11 for linefill 1 evictions.

8.1.4 AXI USER bits

The AXI USER bits encodings are as follows:

Data side read bus, **ARUSERM0[6:0]**

Table 8-3 shows the bit encodings for **ARUSERM0[6:0]**

Table 8-3 ARUSERM0[6:0] encodings

Bits	Name	Description
[6]	Reserved	b0
[5]	L2 Prefetch hint	Indicates that the read access is a prefetch hint to the L2, and does not expect any data back
[4:1]	Inner attributes	b0000 = Strongly Ordered b0001 = Device b0011 = Normal Memory Non-Cacheable b0110 = Write-Through b0111 = Write-Back no Write-Allocate b1111 = Write-Back Write-Allocate.
[0]	Shared bit	0 = Nonshared 1 = Shared.

Instruction side read bus, **ARUSERM1[6:0]**

Table 8-4 shows the bit encodings for **ARUSERM1[6:0]**.

Table 8-4 ARUSERM1[6:0] encodings

Bits	Name	Description
[6]	Reserved	b0

Table 8-4 ARUSERM1[6:0] encodings (continued)

Bits	Name	Description
[5]	Reserved	b0
[4:1]	Inner attributes	b0000 = Strongly Ordered b0001 = Device b0011 = Normal Memory Non-Cacheable b0110 = Write-Through b0111 = Write-Back no Write-Allocate b1111 = Write-Back Write-Allocate.
[0]	Shared bit	0 = Nonshared 1 = Shared.

Data side write bus, AWUSERM0[8:0]

Table 8-5 shows the bit encodings for **AWUSERM0[8:0]**.

Table 8-5 AWUSERM0[8:0] encodings

Bits	Name	Description
[8]	Early BRESP Enable bit	Indicates that the L2 slave can send an early BRESP answer to the write request. See *Early BRESP* on page 8-7.
[7]	Write full line of zeros bit	Indicates that the access is an entire cache line write full of zeros. See *Write full line of zeros* on page 8-8.
[6]	Clean eviction	Indicates that the write access is the eviction of a clean cache line.
[5]	L1 eviction	Indicates that the write access is a cache line eviction from the L1.
[4:1]	Inner attributes	b0000 = Strongly Ordered b0001 = Device b0011 = Normal Memory Non-Cacheable b0110 = Write-Through b0111 = Write-Back no Write-Allocate b1111 = Write-Back Write-Allocate.
[0]	Shared bit	0 = Nonshared 1 = Shared.

8.1.5 Exclusive L2 cache

The Cortex-A9 processor can be connected to an L2 cache that supports an exclusive cache mode. This mode must be activated both in the Cortex-A9 processor and in the L2 cache controller.

In this mode, the data cache of the Cortex-A9 processor and the L2 cache are exclusive. At any time, a given address is cached in either L1 data caches or in the L2 cache, but not in both. This has the effect of greatly increasing the usable space and efficiency of an L2 cache connected to the Cortex-A9 processor. When exclusive cache configuration is selected:

- Data cache line replacement policy is modified so that the victim line always gets evicted to L2 memory, even if it is clean.

- If a line is dirty in the L2 cache controller, a read request to this address from the processor causes writeback to external memory and a linefill to the processor.

8.2 Optimized accesses to the L2 memory interface

This section describes optimized accesses to the L2 memory interface. These optimized accesses can generate non-AXI compliant requests on the Cortex-A9 AXI master ports. These non-AXI compliant requests must be generated only when the slaves connected on the Cortex-A9 AXI master ports can support them. The L2 cache controller supports these kinds of requests. The following subsections describe the requests:

- *Prefetch hint to the L2 memory interface*
- *Early BRESP*
- *Write full line of zeros* on page 8-8
- *Speculative coherent requests* on page 8-8.

8.2.1 Prefetch hint to the L2 memory interface

The Cortex-A9 processor can generate prefetch hint requests to the L2 memory controller. The prefetch hint requests are non-compliant AXI read requests generated by the Cortex-A9 processor that do not expect any data return.

You can generate prefetch hint requests to the L2 by:

- Enabling the L2 Prefetch Hint feature, bit [1] in the ACTLR. When enabled, this feature enables the Cortex-A9 processor to automatically issue L2 prefetch hint requests when it detects regular fetch patterns on a coherent memory. This feature is only triggered in a Cortex-A9 MPCore processor, and not in a uniprocessor.
- Programming PLE operations, when this feature is available in the Cortex-A9 processor. In this case, the PLE engine issues a series of L2 prefetch hint requests at the programmed addresses. See Chapter 9 *Preload Engine*.

L2 prefetch hint requests are identified by having their ARUSER[5] bit set.

Note

No additional programming of the L2C-310 is required.

8.2.2 Early BRESP

BRESP answers on response channels must be returned to the master only when the last data has been sent by the master. The Cortex-A9 processor can also deal with **BRESP** answers returned as soon as address has been accepted by the slave, regardless of whether data is sent or not. This enables the Cortex-A9 processor to provide a higher bandwidth for writes if the slave can support the Early BRESP feature. The Cortex-A9 processor sets the **AWUSER[8]** bit to indicate to the slave that it can accept an early **BRESP** answer for this access. This feature can optimize the performance of the processor, but the Early **BRESP** feature generates non-AXI 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, AXI compliant, or in advance, non-AXI compliant. The L2C-310 cache controller supports this non-AXI compliant feature.

The Cortex-A9 processor does not require any programming to enable this feature, that is always on by default.

Note

You must program the L2 cache controller to benefit from this optimization. See the *CoreLink Level 2 Cache Controller (L2C-310) Technical Reference Manual*.

8.2.3 Write full line of zeros

When this feature is enabled, the Cortex-A9 processor can write entire non-coherent cache lines full of zero to the L2C-310 cache controller with a single request. This provides a performance improvement and some power savings. This feature can optimize the performance of the processor, but it requires a slave that is optimized for this special access. The requests are marked as write full line of zeros by having the associated **AWUSERM0[7]** bit set.

Setting bit [3] of the ACTLR enables this feature. See *Auxiliary Control Register* on page 4-27.

You must program the L2C-310 Cache Controller first, prior to enabling the feature in the Cortex-A9 processor, to support this feature. See the *CoreLink Level 2 Cache Controller (L2C-310) Technical Reference Manual*.

8.2.4 Speculative coherent requests

This optimization is available for Cortex-A9 MPCore processors only. See the *Cortex-A9 MPCore TRM*.

8.3 STRT instructions

Take particular care with noncacheable 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 and after the `STRT`.

  This prevents an `STRT` from merging into an existing slot at the same 64-bit address, or merging with another write at the same 64-bit address.

Table 8-6 shows Cortex-A9 modes and corresponding **AxPROT** values.

Table 8-6 Cortex-A9 mode and AxPROT values

Processor mode	Type of access	Value of AxPROT
User	Cacheable read access	User
Privileged		Privileged
User	Noncacheable read access	User
Privileged		Privileged
-	Cacheable write access	Always marked as Privileged
User	Noncacheable write access	User
Privileged	Noncacheable write access	Privileged, except when using `STRT`

Chapter 9
Preload Engine

The design can include a *Preload Engine* (PLE). The PLE loads selected regions of memory into the L2 interface. This chapter describes the PLE. It contains the following sections:

- *About the Preload Engine* on page 9-2
- *PLE control register descriptions* on page 9-3
- *PLE operations* on page 9-4.

9.1 About the Preload Engine

If implemented, the PLE loads selected regions of memory into the L2 interface. Use the MCRR preload channel operation to program the PLE. Dedicated events monitor the behavior of the memory region. Additional L2C-310 events can also monitor PLE behavior.

The preload operation parameters enter the PLE FIFO that includes:

- programmed parameters:
  - base address
  - length of stride
  - number of blocks.
- a valid bit
- an NS state bit
- a *Translation Table Base* (TTB) address
- an *Address Space Identifier* (ASID) value.

Preload blocks can span multiple page entries. Programmed entries can still be valid in case of context switches.

The Preload Engine handles cache line preload requests in the same way as a standard PLD request except that it uses its own TTB and ASID parameters. If there is a translation abort, the preload request is ignored and the Preload Engine issues the next request.

Not all the MMU settings are saved. The Domain, Tex-Remap, Primary Remap, Normal Remap, and Access Permission registers are not saved. As a consequence, a write operation in any of these registers causes a flush of the entire FIFO and of the active channel. Additionally, for TLB maintenance operations, the maintenance operation must also be applied to the FIFO entries. This is done as follows:

On Invalidate by MVA and ASID

Invalidate all entries with a matching ASID.

On Invalidate by ASID

Invalidate all entries with a matching ASID.

On Invalidate by MVA all ASID

Flush the entire FIFO.

On Invalidate entire TLB

Flush the entire FIFO.

.

These rules are also applicable to the PLE active channel.

The Preload Engine defines the following `MCRR` instruction to use with the preload blocks.

```
MCRR p15, 0, <Rt>,<Rt2> c11; Program new PLE channel
```

The number of entries in the FIFO can be set as an RTL configuration design choice. Available sizes are:

- 16 entries
- 8 entries
- 4 entries.

9.2 PLE control register descriptions

The PLE control registers are CP15 registers, accessed when CRn is c11. See *c11 registers* on page 4-10. The following sections describe the PLE control registers:

- *PLE ID Register* on page 4-36
- *PLE Activity Status Register* on page 4-36
- *PLE FIFO Status Register* on page 4-37
- *Preload Engine User Accessibility Register* on page 4-38
- *Preload Engine Parameters Control Register* on page 4-39.

For all CP15 c11 system control registers, NSAC.PLE controls Non-secure accesses. *PLE operations* on page 9-4 shows the operations to use with these control registers.

9.3 PLE operations

The following sections describe the PLE operations:

- *Preload Engine FIFO flush operation*
- *Preload Engine pause channel operation*
- *Preload Engine resume channel operation*
- *Preload Engine kill channel operation* on page 9-5
- *PLE Program New Channel operation* on page 9-5.

For all Preload Engine operations:

- NSACR.PLE controls Non-secure execution.
- PLEUAR.EN controls User execution.
- the operations are only available in configurations where the Preload Engine is present, otherwise an Undefined Instruction exception is taken.

9.3.1 Preload Engine FIFO flush operation

The PLEFF operation characteristics are:

Purpose	Flushes all PLE channels programmed previously including the PLE channel being executed.

To perform the PLE FIFO Flush operation, use:

```
MCR p15, 0, <Rt>, c11, c2, 1
```

<Rt> is not taken into account in this operation.

9.3.2 Preload Engine pause channel operation

The PLEPC operation characteristics are:

Purpose	Pauses PLE activity.

You can perform a PLEPC operation even if no PLE channel is active. In this case, even if a new PLE channel is programmed afterwards, its execution does not start until after a PLE Resume Channel operation.

To perform the PLE PC operation, use:

```
MCR p15, 0, <Rt>, c11, c3, 0
```

<Rt> is not taken into account in this operation.

9.3.3 Preload Engine resume channel operation

The PLERC operation characteristics are:

Purpose	Causes Preload Engine activity to resume.

If you perform a PLERC operation when the PLE is not paused, the Resume Channel operation is ignored.

To perform a PLERC operation, use:

```
MCR p15, 0, <Rt>, c11, c3, 1
```

9.3.4 Preload Engine kill channel operation

The PLEKC operation characteristics are:

Purpose Kills the active PLE channel.

This operation does not operate on any PLE request in the PLE FIFO.

To perform a PLEKC operation, use

```
MCR p15, 0, <Rt>, c11, c3, 2
```

9.3.5 PLE Program New Channel operation

The PLE Program new channel operation characteristics are:

Purpose Programs a new memory region to preload into L2 memory. Kills the active PLE channel.

Figure 9-1 shows the <Rt>. and <Rt2> bit assignments for PLE program new channel operations. Rt is the register that contains the Base address. Rt2 is the register that contains the length, stride, and number of blocks.

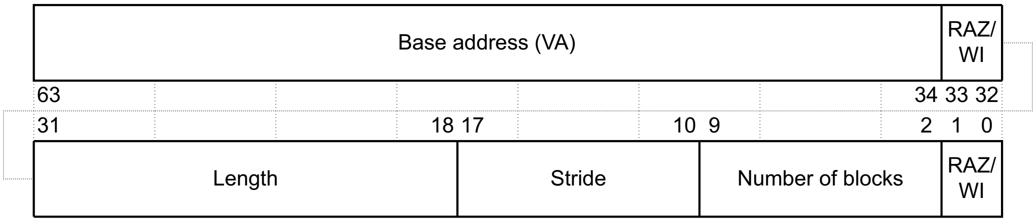


Figure 9-1 Program new channel operation bit assignments

Table 9-1 shows the PLE program new channel operation bit assignments.

Table 9-1 PLE program new channel operation bit assignments

Bits	Name	Description
[63:34]	Base address (VA)	This is the 32-bit Base Virtual Address of the first block of memory to preload. The address is aligned on a word boundary. That is, bits [33:32] are RAZ/WI.
[33:32]	-	RAZ/WI
[31:18]	Length	Specifies the length of the block to preload. Length is encoded as word multiples. The range is from 14’b0000000000, a single word block, to 14’b11111111111111, a 16K word block.
[17:10]	Stride	Indicates the preload stride between blocks. The preload stride is the difference between the start address of two blocks. The stride is encoded as a word multiple. The range is from 8’b00000000, contiguous blocks, to 8’b11111111, prefetch blocks every 256 words.
[9:2]	Number of blocks	Specifies the number of blocks to preload. Values range from 8’b00000000, indicating a single block preload, to 8’b11111111 indicating 256 blocks.
[1:0]	-	RAZ/WI

To program a new channel operation, use the MCRR operation:

```
MCRR p15, 0, <Rt>,<Rt2> c11; Program new PLE channel
```

Note

A newly programmed PLE entry is written to the PLE FIFO if the FIFO has available entries. In cases of FIFO overflow, the instruction silently fails, and the FIFO Overflow event signal is asserted. See Preload events in Table 11-5 on page 11-7. See *PLE FIFO Status Register* on page 4-37.

Chapter 10
Debug

This chapter describes the processor debug unit. This feature assists the development of application software, operating systems, and hardware. This chapter contains the following sections:

10.1 Debug Systems

The Cortex-A9 processor is one component of a debug system. Figure 10-1 shows a typical system.

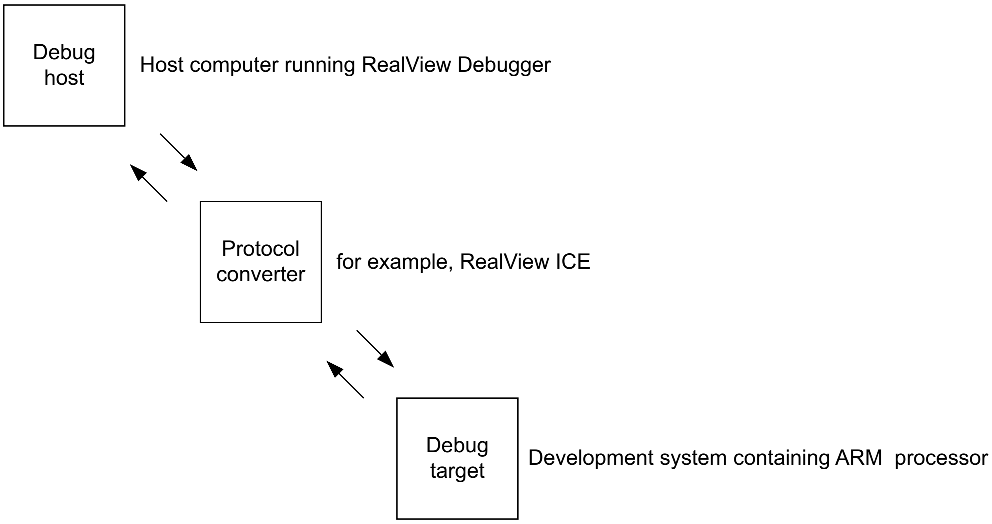


Figure 10-1 Typical debug system

This typical system has three parts:

- *Debug host*
- *Protocol converter*
- *Debug target.*

10.1.1 Debug host

The debug host is a computer, for example a personal computer, running a software debugger such as RealView Debugger. The debug host enables you to issue high-level commands such as setting a breakpoint at a certain location or examining the contents of a memory address.

10.1.2 Protocol converter

The debug host connects to the processor development system using an interface such as Ethernet. The messages broadcast over this connection must be converted to the interface signals of the debug target. A protocol converter performs this function, for example, RealView ICE.

10.1.3 Debug target

The debug target is the lowest level of the system. An example of a debug target is a development system with a Cortex-A9 test chip or a silicon part with a Cortex-A9 processor.

The debug target must implement some system support for the protocol converter to access the processor debug unit using the *Advanced Peripheral Bus* (APB) slave port.

10.2 About the Cortex-A9 debug interface

The Cortex-A9 processor implements the ARMv7 debug architecture as described in the *ARM Architecture Reference Manual*.

In addition, there are:

- Cortex-A9 processor specific events. These are described in *Performance monitoring events* on page 11-7.
- System coherency events.

For more information, see *Performance monitoring* on page 2-3 and Chapter 11 *Performance Monitoring Unit*.

The debug interface consists of:

- a Baseline CP14 interface
- an Extended CP14 interface
- an external debug interface connected to the external debugger through a *Debug Access Port* (DAP).

Figure 10-2 shows the Cortex-A9 debug registers interface.

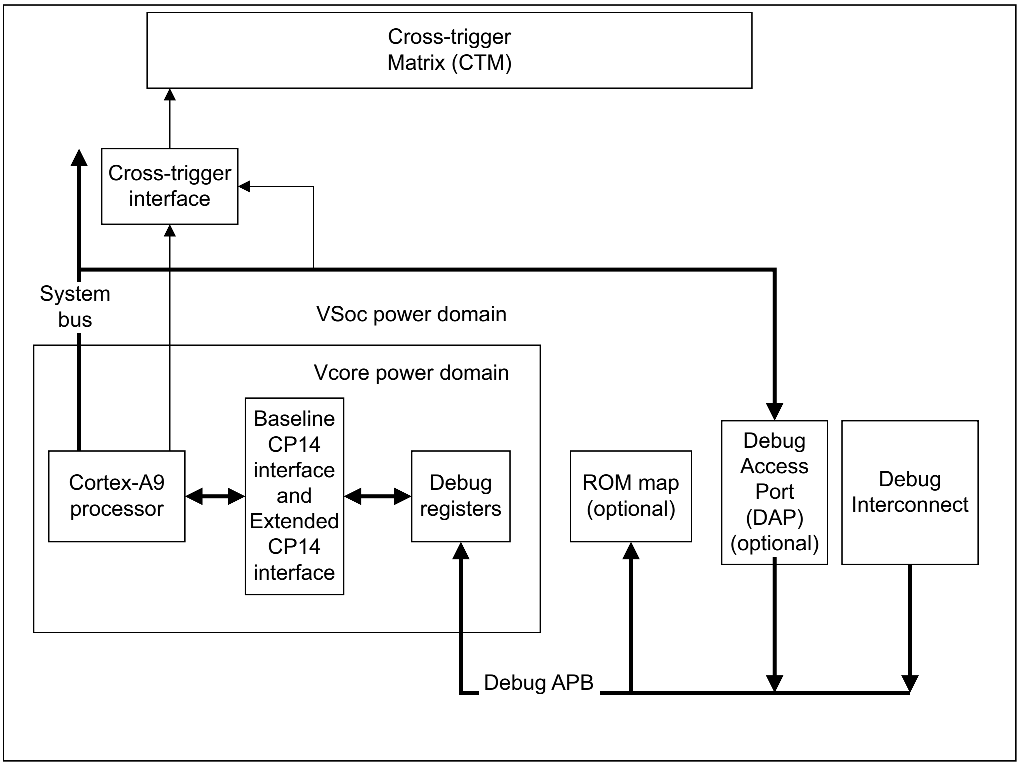


Figure 10-2 Debug registers interface and CoreSight infrastructure

10.3 Debug register features

This section introduces the debug register features in:

- *Processor interfaces*
- *Breakpoints and watchpoints*
- *Effects of resets on debug registers*.

10.3.1 Processor interfaces

The Cortex-A9 processor has the following interfaces to the debug and performance monitor:

Debug registers

This interface is Baseline CP14, Extended CP14, and memory-mapped.

See *CTI signals* on page A-23 and *APB interface signals* on page A-22.

Performance monitor

This interface is CP15 based and memory-mapped. See *Performance monitoring* on page 2-3 and Chapter 11 *Performance Monitoring Unit*.

10.3.2 Breakpoints and watchpoints

The processor supports six breakpoints, two with Context ID comparison, BRP4 and BRP5 and four watchpoints.

See *Breakpoint Value Registers bit functions* on page 10-7 and *BCR Register bit assignments* on page 10-8 for more information on breakpoints.

See *Watchpoint Value Registers bit functions* on page 10-11, *WCR Register bit assignments* on page 10-11 and *Watchpoints* on page 10-15 for more information on watchpoints.

10.3.3 Effects of resets on debug registers

nDBGRESET

This is the debug logic reset signals. This signal must be asserted during a power-on reset sequence. Other reset signals, **nCPURESET** and **nNEONRESET**, if MPE is present, have no effect on the debug logic.

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.

10.4 Debug register summary

You can access the debug registers:

- through the CP14 interface. The debug registers are mapped to coprocessor instructions.
- through the APB using the relevant offset when PADDRDBG[12]=0, with the following exceptions:
  - DBGRAR
  - DBGSAR
  - DBGSCR-int
  - DBGTR-int.

External views of DBSCR and DBGTR are accessible through memory-mapped APB access.

Table 10-1 shows the CP14 interface registers. All other registers are described in the *ARM Architecture Reference Manual*.

Table 10-1 CP14 Debug register summary

Register number	Offset	CRn	Op1	CRm	Op2	Name	Type	Description
0	`0x000`	c0	0	c0	0	DBGDIDR[a][b]	RO	See the *ARM Architecture Reference Manual*
-	-	c1	0	c0	0	DBGDRAR[a]	RO	
-	-	c2	0	c0	0	DBGDSAR[a]	RO	
-	-	c1	0	c1	0	DBGDSCRint[a][b]	RO	
5	-	c0	0	c5	0	DBGDTRRXint[a]	RO	
						DBGDTRTXint[a]	WO	Reserved
6	`0x018`	c0	0	c6	0	DBGWFAR	RW	Use of DBGWFAR is deprecated in the ARMv7 architecture, because watchpoints are synchronous
7	`0x01C`	c0	0	c7	0	DBGVCR	RW	See the *ARM Architecture Reference Manual*
8	-	-	-	-	-	-	-	Reserved
9	`0x024`	c0	0	c9	0	DBGECR	RAZ/WI	Not implemented
10	`0x028`	c0	0	c10	0	DBGDSCCR	RAZ/WI	
11	`0x02C`	c0	0	c11	0	DBGDSMCR	RAZ/WI	
12-31	-	-	-	-	-	-	-	Reserved
32	`0x080`	c0	0	c0	2	DBGDTRRXext	RW	See the *ARM Architecture Reference Manual*
33	`0x084`	c0	0	c1	2	DBGITR	WO	
33	`0x084`	c0	0	c1	2	DBGPCSR	RO	
34	`0x088`	c0	0	c2	2	DBGDSCRext	RW	
35	`0x08C`	c0	0	c3	2	DBGDTRTXext	RW	

Table 10-1 CP14 Debug register summary (continued)

Register number	Offset	CRn	Op1	CRm	Op2	Name	Type	Description
36	`0x090`	c0	0	c4	2	DBGDRCR	WO	See the *ARM Architecture Reference Manual*
37-63	-	-	-	-	-	Reserved	-	Reserved
64-68	`0x100-0x114`	c0	0	c0-c5	4	DBGBVRn	RW	*Breakpoint Value Registers* on page 10-7
69-79	-	-	-	-	-	-	-	Reserved
80-85	`0x140-0x154`	c0	0	c0-c5	5	DBGBCRn	RW	*Breakpoint Control Registers* on page 10-8
86-95	-	-					-	Reserved
96-99	`0x180-0x18C`	c0	0	c0-c3	6	DBGWVRn	RW	*Watchpoint Value Registers* on page 10-10
100-111	-	-	-	-	-	-	-	Reserved
112-115	`0x1C0-0x1CC`	c0	0	c0-c3	7	DBGWCRn	RW	*Watchpoint Control Registers* on page 10-11
116-191	-	-	-	-	-	-	-	Reserved
192	`0x300`	c1	0	c0	4	DBGOSLAR	RAZ/WI	Not implemented
193	`0x304`	c1	0	c1	4	DBGOSLSR	RAZ/WI	
194	`0x308`	c1	0	c2	4	DBGOSSRR	RAZ/WI	
195	-	-	-	-	-	-	-	Reserved
196	`0x310`	c1	0	c4	4	DBGPRCR	RO	See the *ARM Architecture Reference Manual*
197	`0x314`	c1	0	c5	4	DBGPRSR	RO	
198-831	-	-	-	-	-	-	-	Reserved
832-895	`0xD00-0xDFC`	-	-	-	-	Processor ID Registers[c]	RO	*Identification Registers* on page 4-11
896-927	`0xE00-0xE7C`	-	-	-	-	-	-	Reserved
928-959	`0xE80-0xEFC`	c7	0	c0	15, 2-3	-	RAZ/WI	Reserved
960-1023	`0xF00-0xFFC`	-	-	-	-	Debug Management Registers	-	*Debug management registers* on page 10-13

a. Baseline CP14 interface. This register also has an external view through the memory-mapped interface and the CP14 interface.

b. Accessible in User mode if bit [12] of the DBGSCR is clear. Also accessible in privileged modes.

c. The Extended CP14 interface `MRC` and `MCR` instructions that map to these registers are UNDEFINED in User mode and UNPREDICTABLE in privileged modes. You must use the CP15 interface to access these registers.

10.5 Debug register descriptions

This section describes the debug registers.

10.5.1 Breakpoint Value Registers

The *Breakpoint Value Registers* (BVRs) are registers 64-68, at offsets `0x100`-`0x114`. Each BVR is associated with a *Breakpoint Control Register* (BCR), for example:

- BVR0 with BCR0
- BVR1 with BCR1.

This pattern continues up to BVR5 with BCR5.

A pair of breakpoint registers, BVRn and BCRn, is called a *Breakpoint Register Pair* (BRPn).

Table 10-2 shows the BVRs and corresponding BCRs.

Table 10-2 BVRs and corresponding BCRs

Breakpoint Value Registers			Breakpoint Control Registers		
Register number	Offset	Name	Register number	Offset	Name
64	`0x100`	BVR0	80	`0x140`	BCR0
65	`0x104`	BVR1	81	`0x144`	BCR1
66	`0x108`	BVR2	82	`0x148`	BCR2
66	`0x10C`	BVR3	83	`0x14C`	BCR3
67	`0x110`	BVR4	84	`0x150`	BCR4
68	`0x114`	BVR5	85	`0x154`	BCR5

The breakpoint value contained in this register corresponds to either an *Instruction Virtual Address* (IVA) or a context ID. Breakpoints can be set on:

- an IVA
- a context ID value
- an IVA and context ID pair.

For an IVA and context ID pair, two BRPs must be linked. A debug event is generated when both the IVA and the context ID pair match at the same time.

Table 10-3 shows how the bit values correspond with the Breakpoint Value Registers functions.

Table 10-3 Breakpoint Value Registers bit functions

Bits	Name	Description
[31:0]	-	Breakpoint value. The reset value is 0.

Note

- Only BRP4 and BRP5 support context ID comparison.

- BVR0[1:0], BVR1[1:0], BVR2[1:0], and BVR3[1:0] are Should Be Zero or Preserved on writes and Read As Zero 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.

10.5.2 Breakpoint Control Registers

The BCR is a read/write register that contains the necessary control bits for setting:

- breakpoints
- linked breakpoints.

Figure 10-3 shows the BCRs bit assignments.

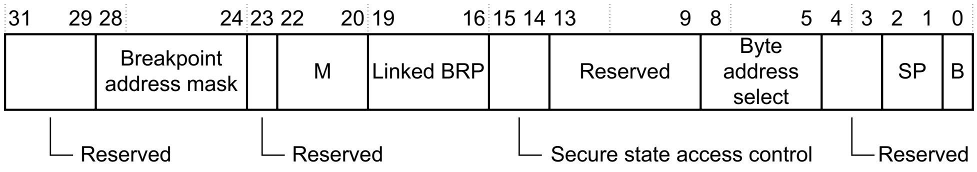


Figure 10-3 BCR Register bit assignments

Table 10-4 shows the BCRs bit assignments.

Table 10-4 BCR Register bit assignments

Bits	Name	Description
[31:29]	-	RAZ on reads, SBZP on writes.
[28:24]	Breakpoint address mask	Breakpoint address mask. RAZ/WI b00000 = no mask.
[23]	-	RAZ on reads, SBZP on writes.
[22:20]	M	Meaning of BVR: b000 = instruction virtual address match b001 = linked instruction virtual address match b010 = unlinked context ID b011 = linked context ID b100 = instruction virtual address mismatch b101 = linked instruction virtual address mismatch b11x = reserved. **Note** BCR0[21], BCR1[21], BCR2[21], and BCR3[21] are RAZ on reads because these registers do not have context ID comparison capability.
[19:16]	Linked BRP	Linked BRP number. The binary number encoded here indicates another BRP to link this one with. **Note** • if a BRP is linked with itself, it is UNPREDICTABLE whether a breakpoint debug event is generated • if this BRP is linked to another BRP that is not configured for linked context ID matching, it is UNPREDICTABLE whether a breakpoint debug event is generated.

Table 10-4 BCR Register bit assignments (continued)

Bits	Name	Description
[15:14]	Secure state access control	Secure state access control. This field enables the breakpoint to be conditional on the security state of the processor: b00 = breakpoint matches in both Secure and Non-secure state b01 = breakpoint only matches in Non-secure state b10 = breakpoint only matches in Secure state b11 = reserved.
[13:9]	-	RAZ on reads, SBZP on writes.
[8:5]	Byte address select	Byte address select. For breakpoints programmed to match an IVA, you must write a word-aligned address to the BVR. You can then use this field to program the breakpoint so it hits only if you access certain byte addresses. If you program the BRP for IVA match, the breakpoint: b0000 = never hits b0011 = hits if any of the two bytes starting at address BVR & `0xFFFFFFFC` +0 is accessed b1100 = hits if any of the two bytes starting at address BVR & `0xFFFFFFFC` +2 is accessed b1111 = hits if any of the four bytes starting at address BVR & `0xFFFFFFFC` +0 is accessed. If you program the BRP for IVA mismatch, the breakpoint hits where the corresponding IVA breakpoint does not hit, that is, the range of addresses covered by an IVA mismatch breakpoint is the negative image of the corresponding IVA breakpoint. If you program the BRP for context ID comparison, this field must be set to b1111. Otherwise, breakpoint and watchpoint debug events might not be generated as expected. **Note** Writing a value to BCR[8:5] where BCR[8] is not equal to BCR[7], or BCR[6] is not equal to BCR[5], has UNPREDICTABLE results.
[4:3]	-	RAZ on reads, SBZP on writes.
[2:1]	SP	Supervisor access control. The breakpoint can be conditioned on the mode of the processor: b00 = User, System, or Supervisor b01 = Privileged b10 = User b11 = any.
[0]	B	Breakpoint enable: 0 = breakpoint disabled, reset value 1 = breakpoint enabled.

Table 10-5 shows the meaning of the BVR as specified by BCR bits [22:20].

Table 10-5 Meaning of BVR as specified by BCR bits [22:20]

BVR[22:20]	Meaning
b000	The corresponding BVR[31:2] is compared against the IVA bus and the state of the processor against this BCR. It generates a breakpoint debug event on a joint IVA and state match.
b001	The corresponding BVR[31:2] is compared against the IVA bus and the state of the processor against this BCR. This BRP is linked with the one indicated by BCR[19:16] linked BRP field. They generate a breakpoint debug event on a joint IVA, context ID, and state match.
b010	The corresponding BVR[31:0] is compared against CP15 Context ID Register, c13 and the state of the processor against this BCR. This BRP is not linked with any other one. It generates a breakpoint debug event on a joint context ID and state match. For this BRP, BCR[8:5] must be set to b1111. Otherwise, it is UNPREDICTABLE whether a breakpoint debug event is generated.
b011	The corresponding BVR[31:0] is compared against CP15 Context ID Register, c13. This BRP links another BRP (of the BCR[21:20]=b01 type), or WRP (with WCR[20]=b1). They generate a breakpoint or watchpoint debug event on a joint IVA or DVA and context ID match. For this BRP, BCR[8:5] must be set to b1111, BCR[15:14] must be set to b00, and BCR[2:1] must be set to b11. Otherwise, it is UNPREDICTABLE whether a breakpoint debug event is generated.
b100	The corresponding BVR[31:2] and BCR[8:5] are compared against the IVA bus and the state of the processor against this BCR. It generates a breakpoint debug event on a joint IVA mismatch and state match.
b101	The corresponding BVR[31:2] and BCR[8:5] are compared against the IVA bus and the state of the processor against this BCR. This BRP is linked with the one indicated by BCR[19:16] linked BRP field. It generates a breakpoint debug event on a joint IVA mismatch, state and context ID match.
b11x	Reserved. The behavior is UNPREDICTABLE.

10.5.3 Watchpoint Value Registers

The *Watchpoint Value Registers* (WVRs) are registers 96-99, at offsets `0x180`-`0x18C`. Each WVR is associated with a *Watchpoint Control Register* (WCR), for example:

- WVR0 with WCR0
- WVR1 with WCR1.

This pattern continues up to WVR3 with WCR3.

Table 10-6 shows the WVRs and corresponding WCRs.

Table 10-6 WVRs and corresponding WCRs

Watchpoint Value Registers			Watchpoint Control Registers		
Register number	Offset	Name	Register number	Offset	Name
96	`0x180`	WVR0	112	`0x1C0`	WCR0
97	`0x184`	WVR1	113	`0x1C4`	WCR1
98	`0x188`	WVR2	114	0x1C8	WCR2
99	`0x18C`	WVR3	115	`0x1DC`	WCR3

A pair of watchpoint registers, WVRn and WCRn, is called a *Watchpoint Register Pair* (WRPn).

The watchpoint value contained in the WVR always corresponds to a *Data Virtual Address* (DVA) and can be set either on:

- a DVA
- a DVA and context ID pair.

For a DVA and context ID pair, a WRP and a BRPs with context ID comparison capability must be linked. A debug event is generated when both the DVA and the context ID pair match simultaneously. Table 10-7 shows how the bit values correspond with the Watchpoint Value Registers functions.

Table 10-7 Watchpoint Value Registers bit functions

Bits	Name	Description
[31:2]	-	Watchpoint address
[1:0]	-	RAZ on reads, SBZP on writes

10.5.4 Watchpoint Control Registers

The WCRs contain the necessary control bits for setting:

- watchpoints
- linked watchpoints.

Figure 10-4 shows the WCRs bit assignments.

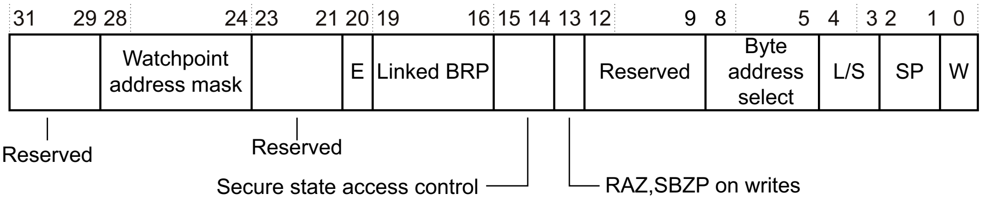


Figure 10-4 WCR Register bit assignments

Table 10-8 shows the WCRs bit assignments.

Table 10-8 WCR Register bit assignments

Bits	Name	Description
[31:29]	-	RAZ on reads, SBZP on writes.
[28:24]	Watchpoint address mask	Watchpoint address mask.
[23:21]	-	RAZ on reads, SBZP on writes.
[20]	E	Enable linking bit: 0 = linking disabled 1 = linking enabled. When this bit is set, this watchpoint is linked with the context ID holding BRP selected by the linked BRP field.
[19:16]	Linked BRP	Linked BRP number. The binary number encoded here indicates a context ID holding BRP to link this WRP with. If this WRP is linked to a BRP that is not configured for linked context ID matching, it is UNPREDICTABLE whether a watchpoint debug event is generated.

Table 10-8 WCR Register bit assignments (continued)

Bits	Name	Description
[15:14]	Secure state access control	Secure state access control. This field enables the watchpoint to be conditioned on the security state of the processor: b00 = watchpoint matches in both Secure and Non-secure state b01 = watchpoint only matches in Non-secure state b10 = watchpoint only matches in Secure state b11 = reserved.
[13]	-	RAZ on reads, SBZP on writes.
[12:9]	-	RAZ/WI.
[8:5]	Byte address select	Byte address select. The WVR is programmed with word-aligned address. You can use this field to program the watchpoint so it only hits if certain byte addresses are accessed.
[4:3]	L/S	Load/store access. The watchpoint can be conditioned to the type of access being done: b00 = reserved b01 = load, load exclusive, or swap b10 = store, store exclusive or swap b11 = either. `SWP` and `SWPB` trigger a watchpoint on b01, b10, or b11. A load exclusive instruction triggers a watchpoint on b01 or b11. A store exclusive instruction triggers a watchpoint on b10 or b11 only if it passes the local monitor within the processor.[a]
[2:1]	SP	Privileged access control. The watchpoint can be conditioned to the privilege of the access being done: b00 = reserved b01 = privileged, match if the processor does a privileged access to memory b10 = user, match only on nonprivileged accesses b11 = either, match all accesses. **Note** For all cases, the match refers to the privilege of the access, not the mode of the processor.
[0]	W	Watchpoint enable: 0 = watchpoint disabled, reset value 1 = watchpoint enabled.

---

[a] A store exclusive can generate an MMU fault or cause the processor to take a data watchpoint exception regardless of the state of the local monitor.

10.6 Debug management registers

The Debug management registers define the standardized set of registers that is implemented by all CoreSight components. This section describes these registers.

You can access these registers:

- through the internal CP14 interface
- through the APB using the relevant offset when PADDRDBG[12]=0

See the *ARM Architecture Reference Manual, ARMv7-A and ARMv7-R edition* for additional information about these registers.

Table 10-9 shows the contents of the management registers for the Cortex-A9 debug unit.

Table 10-9 Debug management registers

Register number	Offset	Name	CRn	Op1	CRm	OP2	Type	Description
960	`0xF00`	DBGITCTRL	c7	0	c0	4	RAZ/ WI	Integration Mode Control Register
961-999	`0xF04-` `0xF9C`	-					RAZ	Reserved
1000	`0xFA0`	DBGCLAIMSET	c7	0	c8	6	RW	Claim Tag Set Register
1001	`0xFA4`	DBGCLAIMCLR	c7	0	c9	6	RW	Claim Tag Clear Register
1002-1003	`0xFA8-` `0xFBC`	-					RAZ	Reserved
1004	`0xFB0`	DBGLAR	c7	0	c12	6	WO	Lock Access Register
	`0xFB4`	DBGLSTR	c7	0	c13	6	RO	Lock Status Register
	`0xFB8`	DBGAUTHSTATUS	c7	0	c14	6	RO	Authentication Status Register
1007-1009	`0xFBC-` `0xFC4`	-					RAZ	Reserved
1010	`0xFC8`	DBGDEVID	c7	0	c1	7	RAZ/ WI	Device ID Register
1011	`0xFCC`	DBGDEVTYPE	c7	0	c3	7	RO	Device Type Register
1012-1023	`0xFD0-` `0xFEC`	DBGPID	c7	0	c4-c8	7	RO	See *Peripheral Identification Registers*.
1020-1023	`0xFF0-` `0xFFC`	DBGCID	c7	0	c12-c15	7	RO	See *Component Identification Registers* on page 10-14

10.6.1 Peripheral Identification Registers

The Peripheral Identification Registers are read-only registers that provide standard information required by all components that conform to the ARM Debug interface v5 specification. The Peripheral Identification Registers are accessible from the Debug APB bus. Only bits [7:0] of each register are used the remaining bits Read-As-Zero. The values in these registers are fixed.

Table 10-10 shows the register number, offset, name, type, value and description that are associated with each Peripheral Identification Register.

Table 10-10 Peripheral Identification Register Summary

Register number	Offset	Name	Type	Value	Description
1012	`0xFD0`	DBGPID4	RO	`0x04`	Peripheral Identification Register 4
1013	`0xFD4`	DBGPID5	RO	-	Reserved
1014	`0xFD8`	DBGPID6	RO	-	Reserved
1015	`0xFDC`	DBGPID7	RO	-	Reserved
1016	`0xFE0`	DBGPID0	RO	`0x09`	Peripheral Identification Register 0
1017	`0xFE4`	DBGPID1	RO	`0xBC`	Peripheral Identification Register 1
1018	`0xFE8`	DBGPID2	RO	`0x0B`	Peripheral Identification Register 2
1019	`0xFEC`	DBGPID3	RO	`0x00`	Peripheral Identification Register 3

See the *ARM Debug Interface v5 Specification* for more information on the Peripheral ID Registers.

10.6.2 Component Identification Registers

The Component Identification Registers are read-only registers that provide standard information required by all components that conform to the ARM Debug interface v5 specification. The Component Identification Registers are accessible from the Debug APB bus. Only bits [7:0] of each register are used the remaining bits Read-As-Zero. The values in these registers are fixed.

Table 10-11 shows the register number, offset, name, type, value and description that are associated with each Component Identification Register.

Table 10-11 Component Identification Register Summary

Register number	Offset	Name	Type	Value	Description
1020	`0xFF0`	DBGCID0	RO	`0x0D`	Component Identification Register 0
1021	`0xFF4`	DBGCID1	RO	`0x90`	Component Identification Register 1
1022	`0xFF8`	DBGCID2	RO	`0x05`	Component Identification Register 2
1023	`0xFFC`	DBGCID3	RO	`0xB1`	Component Identification Register 3

See the *ARM Debug Interface v5 Specification* for more information on the Peripheral ID Registers.

10.7 Debug events

A debug event can be either:

- a software debug event
- a halting debug event.

A processor responds to a debug event in one of the following ways:

- ignores the debug event
- takes a debug exception
- enters debug state.

This section describes debug events in:

- *Watchpoints*
- *Asynchronous aborts*.

10.7.1 Watchpoints

A watchpoint event is always synchronous. It has the same behavior as a synchronous data abort. The method of debug entry, DBGDSCR[5:2], never has the value b0010.

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.

10.7.2 Asynchronous aborts

The Cortex-A9 processor ensures that all possible outstanding asynchronous data aborts are recognized prior to entry to debug state.

10.8 External debug interface

The system can access memory-mapped debug registers through the Cortex-A9 APB slave port.

This APB slave interface supports 32-bits wide data, stalls, slave-generated aborts, and 11 address bits [12:2] mapping 2x4KB of memory. bit [12] of **PADDRDBG[12:0]** selects which of the components is accessed:

- Use **PADDRDBG[12]** = 0 to access the debug area of the Cortex-A9 processor. See Table 10-1 on page 10-5 for debug resources memory mapping.
- Use **PADDRDBG[12]** = 1 to access the PMU area of the Cortex-A9 processor. See Table 11-1 on page 11-3 for PMU resources memory mapping.

The **PADDRDBG31** signal indicates to the processor the source of the access.

See Appendix A *Signal Descriptions* for a complete list of the external debug signals.

Figure 10-5 shows the external debug interface signals.

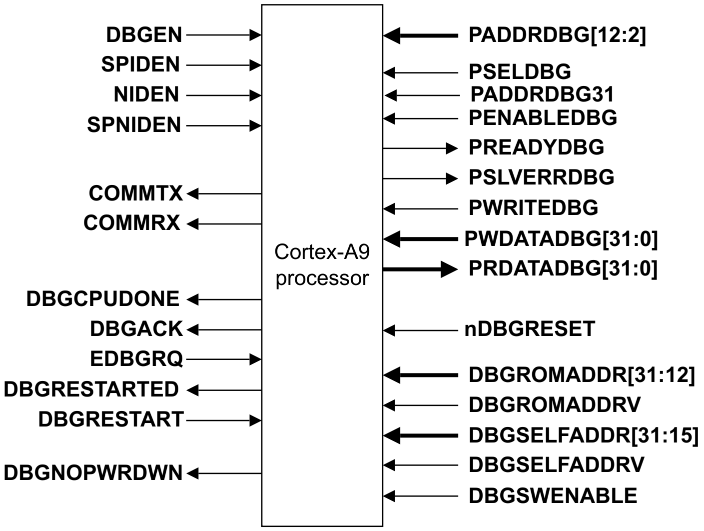


Figure 10-5 External debug interface signals

10.8.1 Debugging modes

Authentication signals control the debugging modes. The authentication signals configure the processor so its activity can only be debugged or traced in a certain subset of processor modes and security states. See *Authentication signals* on page 10-17.

Note

The Cortex-A9 processor only supports halting debug-mode debugging in secure User mode when invasive debugging is enabled by the **SPIDEN** pin. When **SPIDEN** is LOW, only monitor mode debugging in secure User mode is available by setting the SDR.SUIDEN bit. That is, when **SPIDEN** is LOW, the processor is not permitted to enter Halting Debug Mode even if the SDR.SUIDEN bit is set to 1. You can bypass this restriction by setting the external **SPIDEN** pin HIGH.

10.8.2 Authentication signals

Table 10-12 shows a list of the valid combinations of authentication signals along with their associated debug permissions.

Table 10-12 Authentication signal restrictions

SPIDEN	DBGEN[a]	SPNIDEN	NIDEN	Secure[b] invasive debug permitted	Non-secure invasive debug permitted	Secure non-invasive debug permitted	Non-secure non-invasive debug permitted
0	0	0	0	No	No	No	No
0	0	0	1	No	No	No	Yes
0	0	1	0	No	No	No	No
0	0	1	1	No	No	Yes	Yes
0	1	0	0	No	Yes	No	Yes
0	1	0	1	No	Yes	No	Yes
0	1	1	0	No	Yes	Yes	Yes
0	1	1	1	No	Yes	Yes	Yes
1	0	0	0	No	No	No	No
1	0	0	1	No	No	Yes	Yes
1	0	1	0	No	No	No	No
1	0	1	1	No	No	Yes	Yes
1	1	0	0	Yes	Yes	Yes	Yes
1	1	0	1	Yes	Yes	Yes	Yes
1	1	1	0	Yes	Yes	Yes	Yes
1	1	1	1	Yes	Yes	Yes	Yes

[a] When **DBGEN** is LOW, the processor behaves as if DBGDSCR[15:14] equals b00 with the exception that halting debug events are ignored when this signal is LOW.

[b] Invasive debug is defined as those operations that affect the behavior of the processor. For example, taking a breakpoint is defined as invasive debug but performance counters and trace are non-invasive.

10.8.3 Changing the authentication signals

The **NIDEN**, **DBGEN**, **SPIDEN**, and **SPNIDEN** input signals are either tied off to some fixed value or controlled by some external device.

If software running on the Cortex-A9 processor has control over an external device that drives the authentication signals, it must make the change using a safe sequence:

1. Execute an implementation-specific sequence of instructions to change the signal value. For example, this might be a single `STR` instruction that writes certain value to a control register in a system peripheral.

2. If step 1 involves any memory operation, issue a DSB.

3. Poll the DSCR or Authentication Status Register to check whether the processor has already detected the changed value of these signals. This is required because the system might not issue the signal change to the processor until several cycles after the DSB completes.
4. Perform an `ISB`, an Exception entry, or Exception exit.

The software cannot perform debug or analysis operations that depend on the new value of the authentication signals until this procedure is complete. The same rules apply when the debugger has control of the processor through the ITR while in debug state.

The relevant combinations of the **DBGEN**, **NIDEN**, **SPIDEN**, and **SPNIDEN** values can be determined by polling DSCR[17:16], DSCR[15:14], or the Authentication Status Register.

10.8.4 Debug APB Interface

Use the Debug APB interface to access:

- debug registers in Table 10-1 on page 10-5
- debug management registers in Table 10-9 on page 10-13

10.8.5 External debug request interface

The following sections describe the external debug request interface signals:

- *EDBGRQ*
- *DBGACK*
- *DBGCPUDONE*
- *COMMRX and COMMTX* on page 10-19
- *DBGROMADDR, and DBGSELFADDR* on page 10-19.

EDBGRQ

This signal generates a halting debug event, to request the processor to enter debug state. When this occurs, the DSCR[5:2] method of debug entry bits are set to b0100. When **EDBGRQ** is asserted, it must be held until **DBGACK** is asserted. Failure to do so leads to UNPREDICTABLE behavior of the processor.

DBGACK

The processor asserts **DBGACK** to indicate that the system has entered debug state. It serves as a handshake for the **EDBGRQ** signal. The **DBGACK** signal is also driven HIGH when the debugger sets the DSCR[10] DbgAck bit to 1.

DBGCPUDONE

DBGCPUDONE is asserted when the processor has completed a DSB as part of the entry procedure to debug state.

The processor asserts **DBGCPUDONE** only after it has completed all Non-debug state memory accesses. Therefore the system can use **DBGCPUDONE** as an indicator that all memory accesses issued by the processor result from operations performed by a debugger.

Figure 10-6 on page 10-19 shows the Cortex-A9 connections specific to debug request and restart.

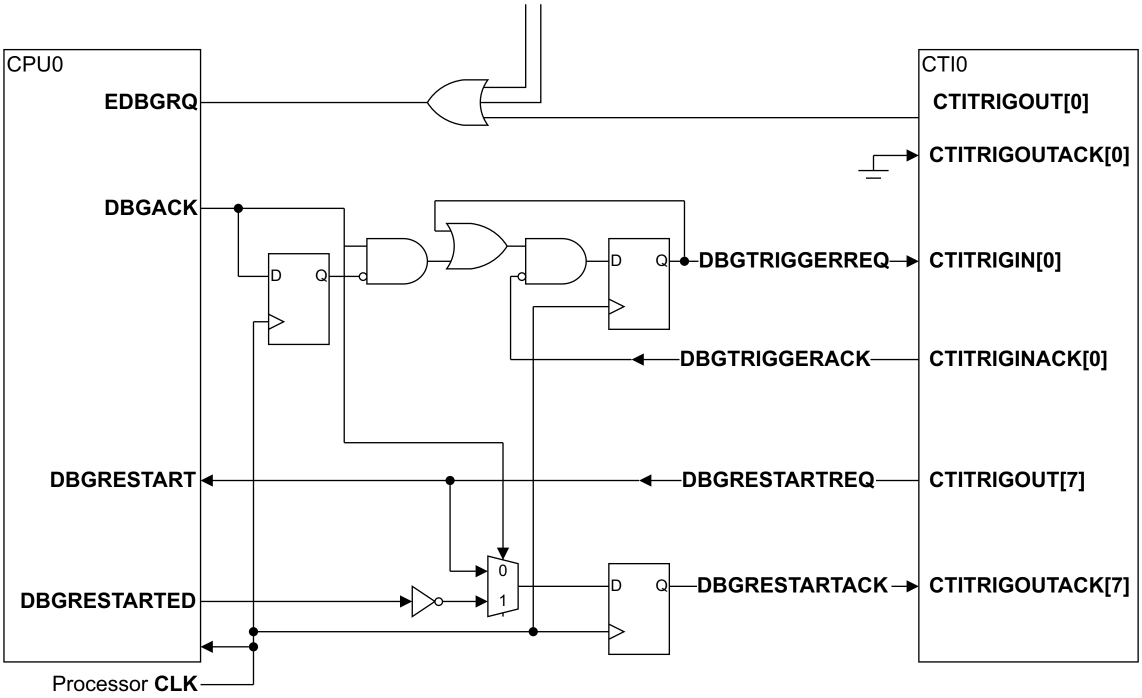


Figure 10-6 Debug request restart-specific connections

COMMRX and COMMTX

The **COMMRX** and **COMMTX** output signals enable interrupt-driven communications over the DTR. By connecting these signals to an interrupt controller, software using the debug communications channel can be interrupted whenever there is new data on the channel or when the channel is clear for transmission.

COMMRX is asserted when the CP14 DTR has data for the processor to read, and it is deasserted when the processor reads the data. Its value is equal to the DBGDSCR[30] DTRRX full flag.

COMMTX is asserted when the CP14 is ready for write data, and it is deasserted when the processor writes the data. Its value ia equal to the inverse of the DBGDSCR[29] DTRTX full flag.

DBGROMADDR, and DBGSELFADDR

The Cortex-A9 processor has a memory-mapped debug interface. The processor can access the debug and PMU registers by executing load and store instructions through the AXI 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 processor registers.

Chapter 11
Performance Monitoring Unit

This chapter describes the *Performance Monitoring Unit* (PMU) and the registers that it uses. It contains the following sections:

11.1 About the Performance Monitoring Unit

The Cortex-A9 PMU provides six counters to gather statistics on the operation of the processor and memory system. Each counter can count any of the 58 events available in the Cortex-A9 processor.

11.2 PMU register summary

You can access the PMU counters, and their associated control registers:

- through the internal CP15 interface
- through the APB, using the relevant offset when PADDRDBG[12]=1.

Table 11-1 gives a summary of the Cortex-A9 PMU registers.

Table 11-1 PMU register summary

Register number	Offset	CRn	Op1	CRm	Op2	Name	Type	Description
0	`0x000`	c9	0	c13	2	PMXEVCNTR0	RW	Event Count Register, see the *ARM Architecture Reference Manual*
1	`0x004`	c9	0	c13	2	PMXEVCNTR1	RW	
2	`0x008`	c9	0	c13	2	PMXEVCNTR2	RW	
3	`0x00C`	c9	0	c13	2	PMXEVCNTR3	RW	
4	`0x010`	c9	0	c13	2	PMXEVCNTR4	RW	
5	`0x014`	c9	0	c13	2	PMXEVCNTR5	RW	
6-30	`0x018-0x078`	-	-	-	-	-	-	Reserved
31	`0x07C`	c9	0	c13	0	PMCCNTR	RW	Cycle Count Register, see the *ARM Architecture Reference Manual*
32-255	`0x080-0x3FC`	-	-	-	-	-	-	Reserved
256	`0x400`	c9	0	c13	1	PMXEVTYPER0	RW	Event Type Selection Register, see the *ARM Architecture Reference Manual*
257	`0x404`	c9	0	c13	1	PMXEVTYPER1	RW	
258	`0x408`	c9	0	c13	1	PMXEVTYPER2	RW	
259	`0x40C`	c9	0	c13	1	PMXEVTYPER3	RW	
260	`0x410`	c9	0	c13	1	PMXEVTYPER4	RW	
261	`0x414`	c9	0	c13	1	PMXEVTYPER5	RW	
262-767	`0x418-0xBFC`	-	-	-	-	-	-	Reserved
768	`0xC00`	c9	0	c12	1	PMCNTENSET	RW	Count Enable Set Register, see the *ARM Architecture Reference Manual*
769-775	`0xC04-0xC1C`	-	-	-	-	-	-	Reserved
776	`0xC20`	c9	0	c12	2	PMCNTENCLR	RW	Count Enable Clear Register, see the *ARM* Architecture Reference Manual
777-783	`0xC24-0xC3C`	-	-	-	-	-	-	Reserved

Table 11-1 PMU register summary (continued)

Register number	Offset	CRn	Op1	CRm	Op2	Name	Type	Description
784	`0xC40`	c9	0	c14	1	PMINTENSET	RW	Interrupt Enable Set Register, see the *ARM Architecture Reference Manual*
785-791	`0xC44-` `0xC5C`	-	-	-	-	-	-	Reserved
792	`0xC60`	c9	0	c14	2	PMINTENCLR	RW	Interrupt Enable Clear Register, see the *ARM Architecture Reference Manual*
793-799	`0xC64-` `0xC7C`	-	-	-	-	-	-	Reserved
800	`0xC80`	c9	0	c12	3	PMOVSR	RW	Overflow Flag Status Register, see the *ARM Architecture Reference Manual*
801-807	`0xC84-` `0xC7C`	-	-	-	-	-	-	Reserved
808	`0xCA0`	c9	0	c12	4	PMSWINC	WO	Software Increment Register, see the *ARM Architecture Reference Manual*
809-831	`0xCA4-` `0xCFC`	-	-	-	-	-	-	Reserved
832-895	-	-	-	-	-	-	-	-
896	`0xE00`	-	-	-	-	-	-	Reserved
897	`0xE04`	c9	0	c12	0	PMCR	RW	Performance Monitor Control Register, see the *ARM Architecture Reference Manual*
898	`0xE08`	c9	0	c14	0	PMUSERENR	RW[a]	User Enable Register, see the *ARM Architecture Reference Manual*
	-	c9	0	c12	5	PMSELR	RW	Event Counter Select Register, see the *ARM Architecture Reference Manual*
899-959	`0xE0C-` `0xEFC`	-	-	-	-	-	-	Reserved
960-1023	`0xF00-` `0xFFC`	-	-	-	-	PMU Management Registers	-	*PMU management registers* on page 11-5

[a] Read-only in User mode.

11.3 PMU management registers

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

You can access these registers through the APB interface only, using the offset listed in Table 11-2 when PADDRDBG[12]=1.

Table 11-2 shows the contents of the management registers for the Cortex-A9 PMU.

Table 11-2 PMU management registers

Register number	Offset	Name	Type	Description
960	`0xF00`	PMITCTRL	RAZ/WI	Integration Mode Control Register
961-999	`0xF04-0xF9C`	-	RAZ	Reserved
1000	`0xFA0`	PMCLAIMSET	RW	Claim Tag Set Register
1001	`0xFA4`	PMCLAIMCLR	RW	Claim Tag Clear Register
1002-1003	`0xFA8-0xFBC`	-	RAZ	Reserved
1004	`0xFB0`	PMLAR	WO	Lock Access Register
1005	`0xFB4`	PMLSR	RO	Lock Status Register
1006	`0xFB8`	PMAUTHSTATUS	RO	Authentication Status Register
1007-1009	`0xFBC-0xFC4`	-	RAZ	Reserved
1010	`0xFC8`	PMDEVID	RAZ/WI	Device ID Register
1011	`0xFCC`	PMDEVTYPE	RO	Device Type Register
1012-1019	`0xFD0-0xFEC`	PMPID	RO	See *Peripheral Identification Registers*
1020- 1023	`0xFF0-0xFFC`	PMCID	RO	See *Component Identification Registers* on page 11-6

11.3.1 Peripheral Identification Registers

The Peripheral Identification Registers are read-only registers that provide standard information required by all components that conform to the ARM Debug interface v5 specification. The Peripheral Identification Registers are accessible from the Debug APB bus. Only bits [7:0] of each register are used the remaining bits Read-As-Zero. The values in these registers are fixed.

Table 11-3 shows the register number, offset value, name, type, value, and description that are associated with each PMU Peripheral Identification Register.

Table 11-3 Peripheral Identification Registers

Register number	Offset	Name	Type	Value	Description
1012	`0xFD0`	PMPID4	RO	`0x04`	Peripheral Identification Register 4
1013	`0xFD4`	PMPID5	RO	-	Reserved
1014	`0xFD8`	PMPID6	RO	-	Reserved
1015	`0xFDC`	PMPID7	RO	-	Reserved

Table 11-3 Peripheral Identification Registers (continued)

Register number	Offset	Name	Type	Value	Description
1016	`0xFE0`	PMPID0	RO	`0xA0`	Peripheral Identification Register 0
1017	`0xFE4`	PMPID1	RO	`0xB9`	Peripheral Identification Register 1
1018	`0xFE8`	PMPID2	RO	`0x0B`	Peripheral Identification Register 2
1019	`0xFEC`	PMPID3	RO	`0x00`	Peripheral Identification Register 3

See the *ARM Debug Interface v5 Specification* for more information on the Peripheral ID Registers.

11.3.2 Component Identification Registers

The Component Identification Registers are read-only registers that provide standard information required by all components that conform to the ARM Debug interface v5 specification. The Component Identification Registers are accessible from the Debug APB bus. Only bits [7:0] of each register are used the remaining bits Read-As-Zero. The values in these registers are fixed.

Table 11-4 shows the offset value, register number, and value that are associated with each PMU Component Identification Register.

Table 11-4 Component Identification Registers

Register number	Offset	Name	Type	Value	Description
1020	`0xFF0`	PMCID0	RO	`0x0D`	Component Identification Register 0
1021	`0xFF4`	PMCID1	RO	`0x90`	Component Identification Register 1
1022	`0xFF8`	PMCID2	RO	`0x05`	Component Identification Register 2
1023	`0xFFC`	PMCID3	RO	`0xB1`	Component Identification Register 3

See the *ARM Debug Interface v5 Specification* for more information on the Component ID Registers.

11.4 Performance monitoring events

The Cortex-A9 processor implements the architectural events described in the *ARM Architecture Reference Manual*, with the exception of:

`0x08`	Instruction architecturally executed.
`0x0E`	Procedure return, other than exception return, architecturally executed.

For events and the corresponding **PMUEVENT** signals, see Table A-18 on page A-14.

The PMU provides an additional set of Cortex-A9 specific events.

11.4.1 Cortex-A9 specific events

Table 11-5 shows the Cortex-A9 specific events. In the value column of Table 11-5 Precise means the event is counted precisely. Events related to stalls and speculative instructions appear as Approximate entries in this column.

Table 11-5 Cortex-A9 specific events

Event	Description	Value
`0x40`	Java bytecode execute[a] Counts the number of Java bytecodes being decoded, including speculative ones.	Approximate
`0x41`	Software Java bytecode executed.[a] Counts the number of software Java bytecodes being decoded, including speculative ones.	Approximate
`0x42`	Jazelle backward branches executed[a]. Counts the number of Jazelle taken branches being executed. This includes the branches that are flushed because of a previous load/store that aborts late.	Approximate
`0x50`	Coherent linefill miss[b] Counts the number of coherent linefill requests performed by the Cortex-A9 processor that also miss in all the other Cortex-A9 processors. This means that the request is sent to the external memory.	Precise
`0x51`	Coherent linefill hit[b] Counts the number of coherent linefill requests performed by the Cortex-A9 processor that hit in another Cortex-A9 processor. This means that the linefill data is fetched directly from the relevant Cortex-A9 cache.	Precise
`0x60`	Instruction cache dependent stall cycles Counts the number of cycles where the processor: • is ready to accept new instructions, • does not receive a new instruction, because: — the instruction side is unable to provide one — the instruction cache is performing at least one linefill.	Approximate
`0x61`	Data cache dependent stall cycles Counts the number of cycles where the processor has some instructions that it cannot issue to any pipeline, and the Load Store unit has at least one pending linefill request, and no pending TLB requests.	Approximate
`0x62`	Main TLB miss stall cycles Counts the number of cycles where the processor is stalled waiting for the completion of translation table walks from the main TLB. The processor stalls because the instruction side is not able to provide the instructions, or the data side is not able to provide the necessary data.	Approximate

Table 11-5 Cortex-A9 specific events (continued)

Event	Description	Value
`0x63`	STREX passed Counts the number of `STREX` instructions architecturally executed and passed.	Precise
`0x64`	STREX failed Counts the number of `STREX` instructions architecturally executed and failed.	Precise
`0x65`	Data eviction Counts the number of eviction requests because of a linefill in the data cache.	Precise
`0x66`	Issue does not dispatch any instruction Counts the number of cycles where the issue stage does not dispatch any instruction because it is empty or cannot dispatch any instructions.	Precise
`0x67`	Issue is empty Counts the number of cycles where the issue stage is empty.	Precise
`0x68`	Instructions coming out of the core renaming stage Counts the number of instructions going through the Register Renaming stage. This number is an approximate number of the total number of instructions speculatively executed, and an even more approximate number of the total number of instructions architecturally executed. The approximation depends mainly on the branch misprediction rate. The renaming stage can handle two instructions in the same cycle so the event is two bits long: b00 = no instructions coming out of the core renaming stage b01 = one instruction coming out of the core renaming stage b10 = two instructions coming out of the core renaming stage.	Approximate
`0x6E`	Predictable function returns Counts the number of procedure returns whose condition codes do not fail, excluding all returns from exception. This count includes procedure returns that are flushed because of a previous load/store that aborts late. Only the following instructions are reported: • `BX R14` • `MOV PC LR` • `POP {..,pc}` • `LDR pc,[sp],#offset.` The following instructions are not reported: • `LDMIA R9!,{..,PC}` (ThumbEE state only) • `LDR PC,[R9],#offset` (ThumbEE state only) • `BX R0 (Rm != R14)` • `MOV PC,R0 (Rm != R14)` • `LDM SP,{...,PC}` (writeback not specified) • `LDR PC,[SP,#offset]` (wrong addressing mode).	Approximate
`0x70`	Main execution unit instructions Counts the number of instructions being executed in the main execution pipeline of the processor, the multiply pipeline and arithmetic logic unit pipeline. The counted instructions are still speculative.	Approximate
`0x71`	Second execution unit instructions Counts the number of instructions being executed in the processor second execution pipeline (ALU). The counted instructions are still speculative.	Approximate

Table 11-5 Cortex-A9 specific events (continued)

Event	Description	Value
`0x72`	Load/Store Instructions Counts the number of instructions being executed in the Load/Store unit. The counted instructions are still speculative.	Approximate
`0x73`	Floating-point instructions Counts the number of floating-point instructions going through the Register Rename stage. Instructions are still speculative in this stage. Two floating-point instructions can be renamed in the same cycle so the event is two bits long: 0b00 = no floating-point instruction renamed 0b01 = one floating-point instruction renamed 0b10 = two floating-point instructions renamed.	Approximate
`0x74`	NEON instructions Counts the number of NEON instructions going through the Register Rename stage. Instructions are still speculative in this stage. Two NEON instructions can be renamed in the same cycle so the event is two bits long: 0b00 = no NEON instruction renamed 0b01 = one NEON instruction renamed 0b10= two NEON instructions renamed.	Approximate
`0x80`	Processor stalls because of PLDs Counts the number of cycles where the processor is stalled because PLD slots are all full.	Approximate
`0x81`	Processor stalled because of a write to memory Counts the number of cycles when the processor is stalled. The data side is stalled also, because it is full and executes writes to the external memory.	Approximate
`0x82`	Processor stalled because of instruction side main TLB miss Counts the number of stall cycles because of main TLB misses on requests issued by the instruction side.	Approximate
`0x83`	Processor stalled because of data side main TLB miss Counts the number of stall cycles because of main TLB misses on requests issued by the data side.	Approximate
`0x84`	Processor stalled because of instruction micro TLB miss Counts the number of stall cycles because of micro TLB misses on the instruction side. This event does not include main TLB miss stall cycles that are already counted in the corresponding main TLB event.	Approximate
`0x85`	Processor stalled because of data micro TLB miss Counts the number of stall cycles because of micro TLB misses on the data side. This event does not include main TLB miss stall cycles that are already counted in the corresponding main TLB event.	Approximate
`0x86`	Processor stalled because of DMB Counts the number of stall cycles because of the execution of a DMB. This includes all `DMB` instructions being executed, even speculatively.	Approximate
`0x8A`	Integer clock enabled Counts the number of cycles when the integer core clock is enabled.	Approximate
`0x8B`	Data engine clock enabled Counts the number of cycles when the data engine clock is enabled.	Approximate

Table 11-5 Cortex-A9 specific events (continued)

Event	Description	Value
`0x90`	`ISB` instructions Counts the number of `ISB` instructions architecturally executed.	Precise
`0x91`	`DSB` instructions Counts the number of `DSB` instructions architecturally executed.	Precise
`0x92`	`DMB` instructions Counts the number of `DMB` instructions speculatively executed.	Approximate
`0x93`	External interrupts Counts the number of external interrupts executed by the processor.	Approximate
`0xA0`	PLE cache line request completed.[c]	Precise
`0xA1`	PLE cache line request skipped.[c]	Precise
`0xA2`	PLE FIFO flush.[c]	Precise
`0xA3`	PLE request completed.[c]	Precise
`0xA4`	PLE FIFO overflow.[c]	Precise
`0xA5`	PLE request programmed.[c]	Precise

[a] Only when the design implements the Jazelle Extension. Otherwise reads as 0.

[b] For use with Cortex-A9 multiprocessor variants.

[c] Active only when the PLE is present. Otherwise reads as 0.

Appendix A
Signal Descriptions

This appendix describes the Cortex-A9 signals. It contains the following sections:

A.1 Clock signals

The Cortex-A9 processor has a single externally generated global clock. Table A-1 shows the clock and clock control signals.

Table A-1 Clock and clock control signals

Name	I/O	Source	Description
CLK	I	Clock controller	Global clock. See *Clocking and resets* on page 2-6.
MAXCLKLATENCY[2:0]	I	Implementation-specific static value	Controls dynamic clock gating delays. This pin is sampled during reset of the processor. See *Power Control Register* on page 4-41.

A.2 Reset signals

Table A-2 shows the reset and reset control signals.

Table A-2 Reset signals

Name	I/O	Source	Description
nCPURESET	I	Reset controller	Cortex-A9 processor reset.
nDBGRESET	I		Cortex-A9 processor debug logic reset.
NEONCLKOFF[a]	I		MPE SIMD logic clock control: 0 = do not cut MPE SIMD logic clock 1 = cut MPE SIMD logic clock.
nNEONRESET[a]	I		Cortex-A9 MPE SIMD logic reset.

a. Only if the MPE is present.

See *Reset* on page 2-6.

A.3 Interrupts

Table A-3 shows the interrupt line signals.

Table A-3 Interrupt line signals

Name	I/O	Source	Description
nFIQ	I	Interrupt sources	Cortex-A9 processor FIQ request input line. Active-LOW fast interrupt request: 0 = activate fast interrupt 1 = do not activate fast interrupt. The processor treats the **nFIQ** input as level sensitive.
nIRQ	I	Interrupt sources	Cortex-A9 processor IRQ request input line. Active-LOW interrupt request: 0 = activate interrupt 1 = do not activate interrupt. The processor treats the **nIRQ** input as level sensitive.

A.4 Configuration signals

Table A-4 shows the configuration signals only sampled during reset of the processor.

Table A-4 Configuration signals

Name	I/O	Source	Description
CFGEND	I	System configuration control	Controls the state of EE bit in the SCTLR at reset: 0 = EE bit is LOW 1 = EE bit is HIGH.
CFGNMFI	I		Configures fast interrupts to be non-maskable: 0 = clear the NMFI bit in the CP15 c1 Control Register 1 = set the NMFI bit in the CP15 c1 Control Register.
TEINIT	I		Default exception handling state: 0 = ARM 1 = Thumb. This signal sets the SCTLR.TE bit at reset.
VINITHI	I		Controls the location of the exception vectors at reset: 0 = start exception vectors at address `0x00000000` 1 = start exception vectors at address `0xFFFF0000`. This signal sets the SCTLR.V bit.

Table A-5 shows the **CP15SDISABLE** signal.

Table A-5 **CP15SDISABLE** signal

Name	I/O	Source	Description
CP15SDISABLE	I	Security controller	Disables write access to some system control processor registers in Secure state: 0 = not enabled 1 = enabled. See *System Control Register* on page 4-24.

A.5 WFE and WFI standby signals

Table A-6 shows the WFE and WFI standby signals.

Table A-6 WFE and WFI standby signals

Name	I/O	Source or destination	Description
EVENTI	I	External coherent agent	Event input for Cortex-A9 processor wake-up from WFE mode.
EVENTO	O		Event output. This signal is active-HIGH for one processor clock cycle when an `SEV` instruction is executed.
STANDBYWFE	O	Power controller	Indicates if the processor is in WFI mode: 0 = processor not in WFI standby mode 1 = processor in WFI standby mode.
STANDBYWFI	O		Indicates if the processor is in WFE mode: 0 = processor not in WFE standby mode 1 = processor in WFE standby mode.

See *Standby modes* on page 2-11.

A.6 Power management signals

Table A-7 shows the power management signals.

Table A-7 Power management signals

Name	I/O	Source	Description
CPURAMCLAMP	I	Power controller	Activates the CPU RAM interface clamps: 0 = clamps not active 1 = clamps active.
NEONCLAMP[a]	I		Activates the Cortex-A9 MPE SIMD logic clamps: 0 = clamps not active 1 = clamps active.

[a] Only if the MPE is present.

See *Power management* on page 2-10.

A.7 AXI interfaces

In Cortex-A9 designs there can be two AXI master ports. The following sections describe the AXI interfaces:

- *AXI Master0 signals data accesses*
- *AXI Master1 signals instruction accesses* on page A-11.

A.7.1 AXI Master0 signals data accesses

The following sections describe the AXI Master0 interface signals used for data read and write accesses:

- *Write address channel signals for AXI Master0*
- *Write data channel signals* on page A-9
- *Write response channel signals* on page A-10
- *Read address channel signals for AXI Master0* on page A-10
- *Read data channel signals* on page A-11
- *AXI Master0 Clock enable signals* on page A-11.

Write address channel signals for AXI Master0

Table A-8 shows the AXI write address channel signals for AXI Master0.

Table A-8 Write address channel signals for AXI Master0

Name	I/O	Source or destination	Description
AWADDRM0[31:0]	O	AXI system devices	Address.
AWBURSTM0[1:0]	O		Burst type = b01, INCR incrementing burst.
AWCACHEM0[3:0]	O		Cache type giving additional information about cacheable characteristics, determined by the memory type and Outer cache policy for the memory region.
AWIDM0[1:0]	O		Request ID.

Table A-8 Write address channel signals for AXI Master0 (continued)

Name	I/O	Source or destination	Description
AWLENM0[3:0]	O	AXI system devices	The number of data transfers that can occur within each burst.
AWLOCKM0[1:0]	O		Lock type.
AWPROTM0[2:0]	O		Protection type.
AWREADYM0	I		Address ready.
AWSIZEM0[1:0]	O		Data transfer size: b000 = 8-bit transfer b001 = 16-bit transfer b010 = 32-bit transfer b011 = 64-bit transfer.
AWUSERM0[8:0]	O		[8] early **BRESP**. Used with L2C-310. [7] write full line of zeros. Used with the L2C-310. [6] clean eviction. [5] level 1 eviction. [4:1] memory type and Inner cache policy. b0000 = Strongly-ordered. b0001 = Device b0011 = Normal Memory Non-Cacheable. b0110 = Write-Through. b0111 = Write-Back no Write-Allocate. b1111 = Write-Back Write-Allocate. [0] shared.
AWVALIDM0	O		Address valid.

Write data channel signals

Table A-9 shows the AXI write data signals for AXI Master0.

Table A-9 AXI-W signals for AXI Master0

Name	I/O	Source or destination	Description
WDATAM0[63:0]	O	AXI system devices	Write data
WIDM0[1:0]	O		Write ID
WLASTM0	O		Write last indication
WREADYM0	I		Write ready
WSTRBM0[7:0]	O		Write byte lane strobe
WVALIDM0	O		Write valid

Write response channel signals

Table A-10 shows the AXI write response channel signals for AXI Master0.

Table A-10 Write response channel signals for AXI Master0

Name	I/O	Source or destination	Description
BIDM0[1:0]	I	AXI system devices	Response ID
BREADYM0	O		Response ready
BRESPM0[1:0]	I		Write response
BVALIDM0	I		Response valid

Read address channel signals for AXI Master0

Table A-11 shows the AXI read address channel signals for AXI Master0.

Table A-11 Read address channel signals for AXI Master0

Name	I/O	Source or destination	Description
ARADDRM0[31:0]	O	AXI system devices	Address.
ARBURSTM0[1:0]	O		Burst type: b01 = INCR incrementing burst b10 = WRAP Wrapping burst.
ARCACHEM0[3:0]	O		Cache type giving additional information about cacheable characteristics.
ARIDM0[1:0]	O		Request ID.
ARLENM0[3:0]	O		The number of data transfers that can occur within each burst.
ARLOCKM0[1:0]	O		Lock type.
ARPROTM0[2:0]	O		Protection type.
ARREADYM0	I		Address ready.
ARSIZEM0[1:0]	O	AXI system devices	Burst size: b000 = 8-bit transfer b001 = 16-bit transfer b010 = 32-bit transfer b011 = 64-bit transfer.
ARUSERM0[4:0]	O		[4:1] memory type and Inner cache policy: b0000 = Strongly-ordered b0001 = Device b0011 = Normal Memory Non-Cacheable b0110 = Write-Through b0111 = Write-Back no Write-Allocate b1111 = Write-Back Write-Allocate. [0] shared.
ARVALIDM0	O		Address valid.

Read data channel signals

Table A-12 shows the AXI read data channel signals for AXI Master0.

Table A-12 Read data channel signals for AXI Master0

Name	I/O	Source or destination	Description
RVALIDM0	I	AXI system devices	Read valid
RDATAM0[63:0]	I		Read data
RRESPM0[1:0]	I		Read response
RLASTM0	I		Read last indication
RIDM0[1:0]	I		Read ID
RREADYM0	O		Read ready

AXI Master0 Clock enable signals

This section describes the AXI Master0 clock enable signals. Table A-13 shows the AXI Master0 clock enable signal.

Table A-13 Clock enable signal for AXI Master0

Name	I/O	Source	Description
ACLKENM0	I	Clock controller	Clock enable for the AXI bus that enables the AXI interface to operate at integer ratios of the system clock. See *Clocking and resets* on page 2-6.

A.7.2 AXI Master1 signals instruction accesses

The following sections describe the AXI Master1 interface signals, that are used for instruction accesses:

- *Read address channel signals for AXI Master1* on page A-12
- *Read data channel signals* on page A-13
- *AXI Master1 Clock enable signals* on page A-13.

Read address channel signals for AXI Master1

Table A-14 shows the AXI read address channel signals for AXI Master1.

Table A-14 Read address channel signals for AXI Master1

Name	I/O	Destination	Description
ARADDRM1[31:0]	O	AXI system devices	Address.
ARBURSTM1[1:0]	O		Burst type: b01 = INCR incrementing burst b10 = WRAP Wrapping burst.
ARCACHEM1[3:0]	O		Cache type giving additional information about cacheable characteristics.
ARIDM1[5:0]	O		Request ID.
ARLENM1[3:0]	O		The number of data transfers that can occur within each burst.
ARLOCKM1[1:0]	O		Lock type: b00 = normal access.
ARPROTM1[2:0]	O		Protection type.
ARREADYM1	I		Address ready.
ARSIZEM1[1:0]	O	AXI system devices	Burst size: b000 = 8-bit transfer b001 = 16-bit transfer b010 = 32-bit transfer b011 = 64-bit transfer.
ARUSERM1[4:0]	O		[4:1] = Inner attributes b0000 = Strongly-ordered b0001 = Device b0011 = Normal Memory Non-Cacheable b0110 = Write-Through b0111 = Write-Back no Write-Allocate b1111 = Write-Back Write-Allocate. [0] = Shared.
ARVALIDM1	O		Address valid.

Read data channel signals

Table A-15 shows the AXI read data signals for AXI Master1.

Table A-15 AXI-R signals for AXI Master1

Name	I/O	Source or destination	Description
RVALIDM1	I	AXI system devices	Read valid
RDATAM1[63:0]	I		Read data
RRESPM1[1:0]	I		Read response
RLASTM1	I		Read last indication
RIDM1[5:0]	I		Read ID
RREADYM1	O		Read ready

AXI Master1 Clock enable signals

Table A-16 shows the AXI Master1 clock enable signals.

Table A-16 Clock enable signal for AXI Master1

Name	I/O	Source	Description
ACLKENM1	I	Clock controller	Clock enable for the AXI bus that enables the AXI interface to operate at integer ratios of the system clock. See *Clocking and resets* on page 2-6.

See Chapter 8 *Level 2 Memory Interface*.

A.8 Performance monitoring signals

Table A-17 shows the performance monitoring signals.

Table A-17 Performance monitoring signals

Name	I/O	Destination	Description
PMUEVENT[57:0]	O	PTM or external monitoring unit	PMU event bus. See Table A-18.
PMUIRQ	O		PMU interrupt signal.
PMUSECURE	O		Gives the status of the Cortex-A9 processor: 0 = in Non-secure state 1 = in Secure state. This signal does not provide input to CoreSight trace delivery infrastructure.
PMUPRIV	O		Gives the status of the Cortex-A9 processor: 0 =in User mode 1 = in privileged mode. This signal does not provide input to CoreSight trace delivery infrastructure.

Table A-18 gives the correlation between **PMUEVENT** signals and their event numbers.

Table A-18 Event signals and event numbers

Name	Event number	Description
PMUEVENT[0]	`0x00`	Software increment
PMUEVENT[1]	`0x01`	Instruction cache miss
PMUEVENT[2]	`0x02`	Instruction micro TLB miss
PMUEVENT[3]	`0x03`	Data cache miss
PMUEVENT[4]	`0x04`	Data cache access
PMUEVENT[5]	`0x05`	Data micro TLB miss
PMUEVENT[6]	`0x06`	Data read
PMUEVENT[7]	`0x07`	Data writes
-	`0x08`	Unused[a]
PMUEVENT[8]	`0x68`	b00 = no instructions renamed b01 = one instruction renamed b10 = two instructions renamed.
PMUEVENT[9]		
PMUEVENT[10]	`0x09`	Exception taken
PMUEVENT[11]	`0x0A`	Exception returns
PMUEVENT[12]	`0x0B`	Write context ID
PMUEVENT[13]	`0x0C`	Software change of PC
PMUEVENT[14]	`0x0D`	Immediate branch
-	`0x0E`	Unused[b]

Table A-18 Event signals and event numbers (continued)

Name	Event number	Description
PMUEVENT[15]	0x6E	Predictable function return[b]
PMUEVENT[16]	0x0F	Unaligned
PMUEVENT[17]	0x10	Branch mispredicted or not predicted
Not exported	0x11	Cycle count
PMUEVENT[18]	0x12	Predictable branches
PMUEVENT[19]	0x40	Java bytecode
PMUEVENT[20]	0x41	Software Java bytecode
PMUEVENT[21]	0x42	Jazelle backward branch
PMUEVENT[22]	0x50	Coherent linefill miss[c]
PMUEVENT[23]	0x51	Coherent linefill hit[c]
PMUEVENT[24]	0x60	Instruction cache dependent stall
PMUEVENT[25]	0x61	Data cache dependent stall
PMUEVENT[26]	0x62	Main TLB miss stall
PMUEVENT[27]	0x63	STREX passed
PMUEVENT[28]	0x64	STREX failed
PMUEVENT[29]	0x65	Data eviction
PMUEVENT[30]	0x66	Issue does not dispatch any instruction
PMUEVENT[31]	0x67	Issue is empty
PMUEVENT[32]	0x70	Main Execution Unit pipe
PMUEVENT[33]	0x71	Second Execution Unit pipe
PMUEVENT[34]	0x72	Load/Store pipe
PMUEVENT[35]	0x73	b00 = no floating-point instruction renamed b01 = one floating-point instruction renamed b10 = two floating-point instructions renamed
PMUEVENT[36]		
PMUEVENT[37]	0x74	b00 = no NEON instruction renamed b01 = one NEON instruction renamed b10 = two NEON instructions renamed
PMUEVENT[38]		
PMUEVENT[39]	0x80	PLD stall
PMUEVENT[40]	0x81	Write stall
PMUEVENT[41]	0x82	Instruction main TLB miss stall
PMUEVENT[42]	0x83	Data main TLB miss stall
PMUEVENT[43]	0x84	Instruction micro TLB miss stall
PMUEVENT[44]	0x85	Data micro TLB miss stall
PMUEVENT[45]	0x86	DMB stall

Table A-18 Event signals and event numbers (continued)

Name	Event number	Description
PMUEVENT[46]	`0x8A`	Integer core clock enabled
PMUEVENT[47]	`0x8B`	Data engine clock enabled
PMUEVENT[48]	`0x90`	ISB
PMUEVENT[49]	`0x91`	DSB
PMUEVENT[50]	`0x92`	DMB
PMUEVENT[51]	`0x93`	External interrupt
PMUEVENT[52]	0xA0	PLE cache line request completed
PMUEVENT[53]	0xA1	PLE cache line request skipped
PMUEVENT[54]	0xA2	PLE FIFO Flush
PMUEVENT[55]	0xA3	PLE request completed
PMUEVENT[56]	0xA4	PLE FIFO Overflow
PMUEVENT[57]	0xA5	PLE request programmed

a. Not generated by Cortex-A9 processors. Replaced by the similar event `0x68`.

b. Not generated by Cortex-A9 processors. Replaced by the similar event `0x6E`.

c. Used in multiprocessor configurations.

See *Cortex-A9 specific events* on page 11-7.

A.9 Exception flags signal

Table A-19 shows the **DEFLAGS** signal.

Table A-19 DEFLAGS signal

Name	I/O	Destination	Description
DEFLAGS[6:0]	O	Exception monitoring unit	Data engine output flags. Only implemented if the Cortex-A9 processor includes a Data engine, either an MPE or FPU. If the DE is MPE: • Bit [6] gives the value of FPSCR[27] • Bit [5] gives the value of FPSCR[7] • Bits [4:0] give the value of FPSCR[4:0]. If the DE is FPU: • Bit [6] is zero. • Bit [5] gives the value of FPSCR[7] • Bits [4:0] give the value of FPSCR[4:0].

For additional information on the FPSCR, see the *Cortex-A9 Floating-Point Unit (FPU) Technical Reference Manual* and the *Cortex-A9 NEON Media Processing Engine Technical Reference Manual*.

A.10 Parity signal

Table A-20 shows the parity signal. This signal is present only if parity is defined. See *Parity error support* on page 7-12.

Table A-20 Parity signal

Name	I/O	Destination	Description
PARITYFAIL[7:0]	O	Parity monitoring device	Parity output pin from the RAM arrays: 0 = no parity fail 1= parity fail Bit [7] BTAC parity error Bit [6] GHB parity error Bit [5] instruction tag RAM parity error Bit [4] instruction data RAM parity error Bit [3] main TLB parity error Bit [2] data outer RAM parity error Bit [1] data tag RAM parity error Bit [0] data data RAM parity error.

A.11 MBIST interface

Table A-21 shows the MBIST interface signals. These signals are present only when the BIST interface is present.

Table A-21 MBIST interface signals

Name	I/O	Source	Description
MBISTADDR[10:0]	I	MBIST controller	MBIST address bus
MBISTARRAY[19:0]	I		MBIST arrays used for testing RAMs
MBISTENABLE	I		MBIST test enable
MBISTWRITEEN	I		Global write enable
MBISTREADEN	I		Global read enable

The size of some MBIST signals depends on whether the implementation has parity support or not. Table A-22 shows these signals with parity support implemented.

Table A-22 MBIST signals with parity support implemented

Name	I/O	Source or destination	Description
MBISTBE[32:0]	I	MBIST controller	MBIST write enable
MBISTINDATA[71:0]	I		MBIST data in
MBISTOUTDATA[71:0]	O		MBIST data out

Table A-23 shows these signals without parity support implemented.

Table A-23 MBIST signals without parity support implemented

Name	I/O	Source/Destination	Description
MBISTBE[25:0]	I	MBIST controller	MBIST write enable
MBISTINDATA[63:0]	I		MBIST data in
MBISTOUTDATA[63:0]	O		MBIST data out

See the *Cortex-A9 MBIST TRM* for a description of MBIST.

A.12 Scan test signal

Table A-24 shows the scan test signal.

Table A-24 Scan test signal

Name	I/O	Destination	Description
SE	I	DFT controller	Scan enable: 0 = not enabled 1 = enabled.

A.13 External Debug interface

The following sections describe the external debug interface signals:

- *Authentication interface*
- *APB interface signals* on page A-22
- *CTI signals* on page A-23
- *Miscellaneous debug interface signals* on page A-23.

A.13.1 Authentication interface

Table A-25 shows the authentication interface signals.

Table A-25 Authentication interface signals

Name	I/O	Source	Description
DBGEN	I	Security controller	Invasive debug enable: 0 = not enabled 1 = enabled.
NIDEN	I		Non-invasive debug enable: 0 = not enabled 1 = enabled.
SPIDEN	I		Secure privileged invasive debug enable: 0 = not enabled 1 = enabled.
SPNIDEN	I		Secure privileged non-invasive debug enable: 0 = not enabled 1 = enabled.

A.13.2 APB interface signals

Table A-26 shows the APB interface signals.

Table A-26 APB interface signals

Name	I/O	Source or destination	Description
PADDRDBG[12:2]	I	CoreSight APB devices	Programming address.
PADDRDBG31	I		APB address bus bit [31]: 0 = not an external debugger access 1 = external debugger access.
PENABLEDBG	I		Indicates a second and subsequent cycle of a transfer.
PSELDBG	I		Debug registers select: 0 = debug registers not selected 1 = debug registers selected.
PWDATADBG[31:0]	I		APB write data.
PWRITEDBG	I		APB read/write signal.
PRDATADBG[31:0]	O		APB read data bus.
PREADYDBG	O		APB slave ready. An APB slave can assert **PREADY** to extend a transfer.
PSLVERRDBG	O		APB slave error signal.

A.13.3 CTI signals

Table A-27 shows the CTI signals.

Table A-27 CTI signals

Name	I/O	Source or destination	Description
EDBGRQ	I	External debugger or CoreSight interconnect	External debug request: 0 = no external debug request 1 = external debug request. The processor treats the **EDBGRQ** input as level sensitive. The **EDBGRQ** input must be asserted until the processor asserts **DBGACK**.
DBGACK	O		Debug acknowledge signal.
DBGCPUDONE	O		Indicates that all memory accesses issued by the Cortex-A9 processor result from operations that are performed by a debugger. active-HIGH.
DBGRESTART	I		Causes the processor to exit from Debug state. It must be held HIGH until **DBGRESTARTED** is deasserted. 0 = not enabled 1 = enabled.
DBGRESTARTED	O		Used with **DBGRESTART** to move between Debug state and Normal state. 0 = not enabled 1 = enabled.

A.13.4 Miscellaneous debug interface signals

Table A-28 shows the miscellaneous debug interface signals.

Table A-28 Miscellaneous debug signals

Name	I/O	Source or destination	Description
COMMRX	O	Debug comms channel	Communications channel receive. Receive portion of Data Transfer Register full flag: 0 = empty 1 = full.
COMMTX	O	Debug comms channel	Communications channel transmit. Transmit portion of Data Transfer Register full flag: 0 = empty 1 = full.
DBGNOPWRDWN	O	Debugger	The debugger has requested that the Cortex-A9 processor is not powered down.
DBGSWENABLE	I	External debugger	When LOW only the external debug agent can modify the debug registers. 0 = not enabled. 1 = enabled.

Table A-28 Miscellaneous debug signals (continued)

Name	I/O	Source or destination	Description
DBGROMADDR[31:12]	I	System configuration	Specifies bits [31:12] of the ROM table physical address. If the address cannot be determined tie this signal LOW.
DBGROMADDRV	I		Valid signal for **DBGROMADDR**. If the address cannot be determined tie this signal LOW.
DBGSELFADDR[31:15]	I		Specifies bits [31:15] of the two's complement signed offset from the ROM table physical address to the physical address where the debug registers are memory-mapped. If the offset cannot be determined tie this signal LOW.
DBGSELFADDRV	I		Valid signal for **DBGSELFADDR**. If the offset cannot be determined tie this signal LOW.

See Chapter 10 *Debug*.

A.14 PTM interface signals

Table A-29 shows the PTM interface signals. These signals are present only if the PTM interface is present.

In the I/O column, the I indicates an input from the PTM interface to the Cortex-A9 processor. The O indicates an output from the Cortex-A9 processor to the PTM. All these signals are in the Cortex-A9 clock domain.

Table A-29 PTM interface signals

Name	I/O	Source or destination	Description
WPTCOMMIT[1:0]	O	PTM device	Number of waypoints committed in this cycle. It is valid to indicate a valid waypoint and commit it in the same cycle.
WPTCONTEXTID[31:0]	O		Context ID for the waypoint. This signal must be true regardless of the condition code of the waypoint. If the processor Context ID has not been set, then **WPTCONTEXTID[31:0]** must report 0.
WPTENABLE	I		Enable waypoint.
WPTEXCEPTIONTYPE[3:0]	O		Exception type: b0001 = Halting debug-mode b0010 = Secure Monitor b0100 = Imprecise Data Abort b0101 = T2EE trap b1000 = Reset b1001 = UNDEF b1010 = SVC b1011 = Prefetch abort/software breakpoint b1100 = Precise data abort/software watchpoint b1110 = IRQ b1111 = FIQ.
WPTFLUSH	O		Waypoint flush signal.
WPTLINK	O		The waypoint is a branch that updates the link register. Only HIGH if **WPTTYPE** is a direct branch or an indirect branch.

Table A-29 PTM interface signals (continued)

Name	I/O	Source or destination	Description
WPTPC[31:0]	O	PTM device	Waypoint last executed address indicator. This is the base Link Register in the case of an exception. Equal to 0 if the waypoint is a reset exception.
WPTT32LINK	O		Indicates the size of the last executed address when in Thumb state: 0 = 16-bit instruction 1 = 32-bit instruction.
WPTTAKEN	O		The waypoint passed its condition codes. The address is still used, irrespective of the value of this signal. Must be set for all waypoints except branch.
WPTTARGETJBIT	O		J bit for waypoint destination.
WPTTARGETPC[31:0]	O		Waypoint target address. Bit [1] must be zero if the T bit is zero. Bit [0] must be zero if the J bit is zero. The value is zero if **WPTTYPE** is either prohibited or debug.
WPTTARGETTBIT	O		T bit for waypoint destination.
WPTTRACEPROHIBITED	O	PTM device	Trace is prohibited for the waypoint target. Indicates entry to prohibited region. No more waypoints are traced until trace can resume. This signal must be permanently asserted if **NIDEN** and **DBGEN** are both LOW, after the in-flight waypoints have exited the processor. Either an exception or a serial branch is required to ensure that changes to the inputs have been sampled. Only one **WPTVALID** cycle must be seen with **WPTTRACEPROHIBITED** set. Trace stops with this waypoint and the next waypoint is an Isync packet. See the *CoreSight PTM Architecture Specification* for a description of the packets used in trace.
WPTTYPE[2:0]	O		Waypoint type. b000 = Direct branch b001 = Indirect branch b010 = Exception b011 = DMB/DSB/ISB b100 = Debug entry b101 = Debug exit b110 = Invalid b111 = Invalid. Debug Entry must be followed by Debug Exit. **Note** Debug exit does not reflect the execution of an instruction.

Table A-29 PTM interface signals (continued)

Name	I/O	Source or destination	Description
WPTVALID	O	PTM device	Waypoint is confirmed as valid.
WPTnSECURE	O		Instructions following the waypoint are executed in Non-secure state. An instruction is in Non-secure state if the NS bit is set and the processor is not in secure monitor mode. See *About system control* on page 4-2 for information about Security Extensions.
WPTFIFOEMPTY	O		There are no speculative waypoints in the PTM interface FIFO.

See *Interfaces* on page 2-4.

Appendix B
Cycle Timings and Interlock Behavior

This chapter describes the cycle timings of integer instructions on Cortex-A9 processors. It contains the following sections:

B.1 About instruction cycle timing

This chapter provides information to estimate how much execution time particular code sequences require. The complexity of the Cortex-A9 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. Detailed descriptions of all possible instruction interactions, and all possible events taking place in the processor, is beyond the scope of this document.

B.2 Data-processing instructions

Table B-1shows the execution unit cycle time for data-processing instructions.

Table B-1 shows the following cases:

no shift on source registers

For example, `ADD r0, r1, r2`

shift by immediate source register

For example, `ADD r0, r1, r2 LSL #2`

shift by register

For example, `ADD r0, r1, r2 LSL r3`.

Table B-1 Data-processing instructions cycle timings

Instruction	No shift	Shift by	
		Constant	Register
`MOV`	1	1	2
`AND`, `EOR`, `SUB`, `RSB`, `ADD`, `ADC`, `SBC`, `RSC`, `CMN`, `ORR`, `BIC`, `MVN`, `TST`, `TEQ`, `CMP`	1	2	3
`QADD`, `QSUB`,`QADD8`, `QADD16`, `QSUB8`, `QSUB16`, `SHADD8`, `SHADD16`, `SHSUB8`, `SHSUB16`,`UQADD8`, `UQADD16`, `UQSUB8`, `UQSUB16`,`UHADD8`, `UHADD16`, `UHSUB8`, `UHSUB16`,`QASX`, `QSAX`, `SHASX`, `SHSAX`,`UQASX`, `UQSAX`, `UHASX`, `UHSAX`	2	-	-
`QDADD`, `QDSUB`, `SSAT`, `USAT`	3	-	-
`PKHBT`, `PKHTB`	1	2	-
`SSAT16`, `USAT16`, `SADD8`, `SADD16`, `SSUB8`, `SSUB16`,`UADD8`, `UADD16`, `USUB8`, `USUB16`, `SASX`, `SSAX`, `UASX`, `USAX`	1	-	-
`SXTAB`, `SXTAB16`, `SXTAH`, `UXTAB`, `UXTAB16`, `UXTAH`	3	-	-
`SXTB`, `STXB16`, `SXTH`, `UXTB`, `UTXB16`, `UXTH`	2	-	-
`BFC`, `BFI`, `UBFX`, `SBFX`	2	-	-
`CLZ`, `MOVT`, `MOVW`, `RBIT`, `REV`, `REV16`, `REVSH`, `MRS`	1	-	-
`MSR` not modifying mode or control bits. See *Serializing instructions* on page B-9.	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-A9 processor has special paths that immediately forward data from a load instruction to a subsequent data processing instruction in the execution units.

This path is used when the following conditions are met:

- the data-processing instruction is one of: `SUB`, `RSB`, `ADD`, `ADC`, `SBC`, `RSC`, `CMN`, `MVN`, or `CMP`
- the forwarded source register is not part of a shift operation.

Table B-2 shows cycle timing for single load and store operations. The result latency is the latency of the first loaded register.

Table B-2 Single load and store operation cycle timings

Instruction cycles	AGU cycles	Result latency	
		Fast forward cases	other cases

`LDR ,[reg]`
`LDR ,[reg imm]`
`LDR ,[reg reg]`
`LDR ,[reg reg LSL #2]`
	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]`
	1	3	4

`LDRB` ,[reg] `LDRB` ,[reg imm] `LDRB` ,[reg reg] `LDRB` ,[reg reg LSL #2] `LDRH` ,[reg] `LDRH` ,[reg imm] `LDRH` ,[reg reg] `LDRH` ,[reg reg LSL #2]	2	3	4

`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	4	5

The Cortex-A9 processor can load or store two 32-bit registers in each cycle. However, to access 64 bits, the address must be 64-bit aligned.

This scheduling is done in the *Address Generation Unit* (AGU). The number of cycles required by the AGU to process the load multiple or store multiple operations depends on the length of the register list and the 64-bit alignment of the address. The resulting latency is the latency of the first loaded register. Table B-3 shows the cycle timings for load multiple operations.

Table B-3 Load multiple operations cycle timings

Instruction	AGU cycles to process the instruction Address aligned on a 64-bit boundary		Resulting latency	
	Yes	No	Fast forward case	Other cases
`LDM` ,{1 register}	1	1	2	3

`LDM` ,{2 registers}
`LDRD`
`RFE`
	1	2	2	3

`LDM` ,{3 registers}	2	2	2	3
`LDM` ,{4 registers}	2	3	2	3
`LDM` ,{5 registers}	3	3	2	3
`LDM` ,{6 registers}	3	4	2	3
`LDM` ,{7 registers}	4	4	2	3
`LDM` ,{8 registers}	4	5	2	3
`LDM` ,{9 registers}	5	5	2	3
`LDM` ,{10 registers}	5	6	2	3
`LDM` ,{11 registers}	6	6	2	3
`LDM` ,{12 registers}	6	7	2	3
`LDM` ,{13 registers}	7	7	2	3
`LDM` ,{14 registers}	7	8	2	3
`LDM` ,{15 registers}	8	8	2	3
`LDM` ,{16 registers}	8	9	2	3

Table B-4 shows the cycle timings of store multiple operations.

Table B-4 Store multiple operations cycle timings

Instruction	AGU cycles Aligned on a 64-bit boundary	
	Yes	No
`STM`,{1 register}	1	1

`STM`,{2 registers}
`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
`STM`,{11 registers}	6	6
`STM`,{12 registers}	6	7
`STM`,{13 registers}	7	7
`STM`,{14 registers}	7	8
`STM`,{15 registers}	8	8
`STM`,{16 registers}	8	9

B.4 Multiplication instructions

Table B-4 on page B-6 shows the cycle timings for multiplication instructions.

Table B-5 Multiplication instruction cycle timings

Instruction	Cycles	Result latency
`MUL(S)`, `MLA(S)`	2	4
`SMULL(S)`, `UMULL(S)`, `SMLAL(S)`, `UMLAL(S)`	3	4 for the first written register 5 for the second written register
`SMULxy`, `SMLAxy`, `SMULWy`, `SMLAWy`	1	3
`SMLALxy`	2	3 for the first written register 4 for the second written register
`SMUAD`, `SMUADX`, `SMLAD`, `SMLADX`, `SMUSD`, `SMUSDX`, `SMLSD`, `SMLSDX`	1	3
`SMMUL`, `SMMULR`, `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 have different timing characteristics:

- Branch instructions to immediate locations do not consume execution unit cycles.
- Data-processing instructions to the PC register are processed in the execution units as standard instructions. See *Data-processing instructions* on page B-3.
- Load instructions to the PC register are processed in the execution units as standard instructions. See *Load and store instructions* on page B-4.

See *About the L1 instruction side memory system* on page 7-5 for information on dynamic branch prediction.

B.6 Serializing instructions

Out of order execution is not always possible. Some instructions are serializing. Serializing instructions force the processor to complete all modifications to flags and general-purpose registers by previous instructions before the next instruction is executed.

This section describes timings for serializing instructions.

B.6.1 Serializing instructions

The following exception entry instructions are serializing:

- `SVC`
- `SMC`
- `BKPT`
- instructions that take the prefetch abort handler
- instructions that take the Undefined Instruction exception handler.

The following instructions that modify mode or program control are serializing:

- `MSR CPSR` when they modify control or mode bits
- data processing to PC with the S bit set (for example, `MOVS pc, r14`)
- `LDM` pc ^.
- `CPS`
- `SETEND`
- `RFE`.

The following instructions are serializing:

- all `MCR` to CP14 or CP15 except `ISB` and `DMB`
- `MRC` p14 for debug registers
- `WFE`, `WFI`, `SEV`
- `CLREX`
- `DSB`.

In the r1p0 implementation `DMB` waits for all previous `LDR`/`STR` instructions to finish, not for all instructions to finish.

The following instruction, that modifies the SPSR, is serializing:

- `MSR SPSR`.

Appendix C
Revisions

This appendix describes the technical changes between released issues of this book.

Table C-1 Issue A

Change	Location
First release	-

Table C-2 Differences between issue A and issue B

Change	Location
Clarified Load/Store Unit and address generation.	Figure 1-1 on page 1-2.
Changed fast loop mode to small loop mode.	• Figure 1-1 on page 1-2 • Small loop mode on page 1-3 • *Instruction cache features* on page 7-2 • About power consumption control on page 12-6.
Changed branch prediction to dynamic branch prediction.	• *Features* on page 1-6 • *About the L1 instruction side memory system* on page 7-5 • *Branch instructions* on page B-8.
Changed LI cache coherency to L1 data cache coherency.	*Cortex-A9 variants* on page 1-4.
Corrected Processor Feature Register 0 reset value.	Table 4-29 on page 4-46.

Table C-2 Differences between issue A and issue B (continued)

Change	Location
Made PMSWINC descriptions consistent.	• Table 4-29 on page 4-46 • Software Increment Register on page 4-100.
Updated MIDR bits [3:0] from 0 to 1.	Table 4-1 on page 4-5.
Corrected ID_MMFR3 [23:20] bit value to `0x1`.	Table 4-42 on page 4-50.
Corrected AFE bit description.	Table 4-51 on page 4-62.
Corrected Auxiliary Control Register bit field.	• Table 4-52 on page 4-66 • Figure 4-36 on page 4-66.
Corrected S parameter values.	Set/Way format on page 4-83.
Made descriptions of bits [11], [10], and [8] consistent with table.	Figure 4-41 on page 4-87.
Corrected description of event `0x68`, architecturally removed.	Table 4-80 on page 4-123.
Corrected TLB lockdown entries number from 8 to 4.	c10, TLB Lockdown Register on page 4-134.
Corrected A, I, and F bit descriptions.	c12, Interrupt Status Register on page 4-147.
Changed number of micro TLB entries from 8 to 32.	*Micro TLB* on page 6-4.
Removed repeated information about cache types.	*Micro TLB* on page 6-4.
Amended IRGN bits description from TTBCR to TTBR0/TTRBR1.	*Main TLB* on page 6-4.
Added note about invalidating the caches and BTAC before use.	*About the L1 memory system* on page 7-2.
Added parity support scheme information section.	*Parity error support* on page 7-12.
Listed and described L2 master interfaces, M0 and M1.	*Cortex-A9 L2 interface* on page 8-2.
Added cross reference to DBSCR external description. Extended footnote to include reference to the DBSCR external view.	Table 10-1 on page 10-5.
Corrected DBGDSCR description with the addition of internal and external view descriptions.	CP14 c1, Debug Status and Control Register (DBGDSCR) on page 8-9.
Re-ordered and extended MOE bits descriptions.	Table 8-2 on page 8-10.

Table C-2 Differences between issue A and issue B (continued)

Change	Location
Added more cross-references from Table 10-1.	• CP14 c1, Debug Status and Control Register (DBGDSCR) on page 8-9 • Device Power-down and Reset Status Register (DBGPRSR) on page 8-27 • Integration Mode Control Register (DBGITCTRL) on page 8-45 • Claim Tag Clear Register (DBGCLAIMCLR) on page 8-47 • Lock Access Register (DBGLAR) on page 8-48 • Lock Status Register (DBGLSR) on page 8-49 • Authentication Status Register (DBGAUTHSTATUS) on page 8-49 • Device Type Register (DBGDEVTYPE) on page 8-50.
Corrected Table 10-1 footnotes.	Table 10-1 on page 10-5.
Corrected byte address field entries.	Table 10-8 on page 10-11.
Corrected interrupt signal descriptions.	Table A-3 on page A-4.
Extended AXI USER descriptions.	• Table A-8 on page A-8 • Table A-11 on page A-10 • Table A-14 on page A-12.

Table C-3 Differences between issue B and issue C

Change	Location
Removed 2.8.1 LE and BE-8 accesses on a 64-bit wide bus.	-
Removed Chapter 4 Unaligned and Mixed-Endian Data Access Support.	-
Removed the power management signal **BISTSCLAMP**.	-
Added dynamic high level clock gating.	Dynamic high level clock gating on page 2-9
Updated TLB information.	Table 1-1 on page 1-10, Table 4-10 on page 4-15, Table 4-37 on page 4-44
Shortened ID_MMF3[15:12] description.	Memory Model Features Register 3 on page 4-49
Updated ACTLR to include reference to PL310 optimizations.	Auxiliary Control Register on page 4-64
Added information about a second replacement strategy. Selection done by SCTLR.RR bit.	*System Control Register* on page 4-24
Extended event information.	Cortex-A9 specific events on page 4-32
Added **DEFLAGS[6:0]**.	DEFLAGS[6:0] on page 4-37, *Performance monitoring signals* on page A-14
Added Power Control Register description.	Power Control Register on page 4-63
Added PL310 optimizations to L2 memory interface description.	*Optimized accesses to the L2 memory interface* on page 8-7

Table C-3 Differences between issue B and issue C (continued)

Change	Location
Added watchpoint address masking.	*Watchpoint Control Registers* on page 10-11
Added debug request restart diagram.	*Effects of resets on debug registers* on page 10-4
Added **CPUCLKOFF** information.	Table A-4 on page A-5,Unregistered signals on page B-3
Added **DECLKOFF** information.	Table A-4 on page A-5,Unregistered signals on page B-3
Added **MAXCLKLATENCY[2:0]** information.	*Configuration signals* on page A-5
Extended **PMUEVENT** bus description.	*Performance monitoring signals* on page A-14
Added **PMUSECURE** and **PMUPRIV.**	*Performance monitoring signals* on page A-14
Updated description of serializing behavior of `DMB`.	*Serializing instructions* on page B-9

Table C-4 Differences between issue C and issue D

Change	Location
Included *Preload Engine* (PE) in block diagram	Figure 1-1 on page 1-2
Amended interrupt signals	
Clarified data engine options	*Data engine* on page 1-2
Clarified system design components	*System design components* on page 1-3
Clarified Compliance	*Compliance* on page 1-5
Added PE to features	*Features* on page 1-6
Included PE and PE FIFO size in configurable options	*Configurable options* on page 1-8
Clarified NEON SIMD and FPU options	Table 1-1 on page 1-8
Added *Test Features* section	*Test features* on page 1-9
Reworded the PTM interface section	*Performance monitoring* on page 2-3
Added a new section for Virtualization of interrupts	*Virtualization of interrupts* on page 2-3
Included NEON SIMD clock gating in power control description	*Power Control Register* on page 2-9
Replaced **nDERESET** with **nNEONRESET**	*Reset modes* on page 2-7
Added **nWDRESET**	
Added **nPERIPHRESET**	
Changed voltage domain boundaries and description	Figure 2-4 on page 2-14
2.5.4 Date Engine logic reset replaced	*MPE SIMD logic reset* on page 2-8
Cortex-A9 input signals **DECLAMP** removed, level shifters reference removed	*Communication to the power management controller* on page 2-13
Table 3-1 J and T bit encoding removed	-
The Jazelle extension on page 3-3 moved	*The Jazelle Extension* on page 3-4
NEON technology on page 3-4 renamed and rewritten	*Advanced SIMD architecture* on page 3-5

Table C-4 Differences between issue C and issue D (continued)

Change	Location
3.4 Processor operating states removed	-
3.5 Data types removed	-
Multiprocessing Extensions section added	*Multiprocessing Extensions* on page 3-7
3.6 Memory formats renamed and moved	*Memory model* on page 3-9
3.8 Security Extensions overview renamed and moved	*Security Extensions architecture* on page 3-6
Removed content, tables and figures from 4.1 that duplicates *ARM Architecture Reference Manual* material	*About system control* on page 4-2
4.2 Duplicates of *ARM Architecture Reference Manual* material removed, section renamed	*Register summary* on page 4-3
4.3 Duplicates of *ARM Architecture Reference Manual* material removed, section renamed	*Register descriptions* on page 4-18
Footnote e removed	Table 4-3 on page 4-6
Preload Engine registers added	*c11 registers* on page 4-10
-	*PLE ID Register* on page 4-36
-	*PLE Activity Status Register* on page 4-36
-	*PLE FIFO Status Register* on page 4-37
-	*Preload Engine User Accessibility Register* on page 4-38
-	*Preload Engine Parameters Control Register* on page 4-39
4.4 CP14 Jazelle registers and 4.5 CP14 Jazelle register descriptions in a new chapter	Chapter 5 *Jazelle DBX registers*
Chapter 5 Memory Management Unit, 5.6 MMU software-accessible registers section removed	-
Level 1 Memory System chapter, Cortex-A9 cache policies section removed	-
Prefetch hint to the L2 memory interface, description rewritten and extended	*Prefetch hint to the L2 memory interface* on page 8-7
Clarifications of BRESP and cache controller behavior	*Early BRESP* on page 8-7
Write full line of zeros, signal name corrected to **AWUSERM0[7]**	*Write full line of zeros* on page 8-8
Speculative coherent requests section added	*Speculative coherent requests* on page 8-8
Removed sentence about tying unused bits of **PARITYFAIL** HIGH	*Parity error support* on page 7-12
Added PE description	Chapter 9 *Preload Engine*
Added PMU description	Chapter 11 *Performance Monitoring Unit*
Debug chapter, About debug systems removed	-
Debug chapter, Debugging modes removed	-

Table C-4 Differences between issue C and issue D (continued)

Change	Location
Duplicates of *ARM Architecture Reference Manual* material removed	-
External debug interface, description of **PADDRDBG[12:0]** added	*External debug interface* on page 10-16
Debug APB interface section added	*Debug APB Interface* on page 10-18
Amended and extended signals descriptions, source destination column added	Appendix A *Signal Descriptions*
PMUEVENT[46] description corrected	Table A-17 on page A-14
PMUEVENT[47] description corrected	
Removed AC Characteristics Appendix	-

No differences between issue D and issue E.

Table C-5 Differences between issue D and issue F

Change	Location
PL310 renamed L2C-310	Throughout the book
VFPv3 corrected to VFPv3 D-32	*Media Processing Engine* on page 1-2
Cortex-A9 FPU hardware description rewritten for clarity	*Floating-Point Unit* on page 1-2
SCU description extended	*Cortex-A9 variants* on page 1-4
Dynamic branch prediction description added	*Dynamic branch prediction* on page 2-2
Final paragraph removed	*Energy efficiency features* on page 2-10
WFI/WFE corrected to Standby	Table 2-2 on page 2-10
Renamed and rewritten for clarity	*Standby modes* on page 2-11
Dormant mode clamping information removed	*Dormant mode* on page 2-12
IEM support renamed and rewritten	*Power domains* on page 2-13
Repeated material removed	*About the programmers model* on page 3-2
Debug register description corrected	Table 4-2 on page 4-5
Main ID Register values for r2p1 and r2p2 added	Table 4-2 on page 4-5
Debug register name corrected	Table 4-2 on page 4-5
Descriptions clarified and footnote added.	Table 4-30 on page 4-20
Purpose description extended	*Cache Size Identification Register* on page 4-21
System Control Register value corrected, and footnotes amended	Table 4-3 on page 4-6
Bit [17] function corrected	Table 4-35 on page 4-25
Footnote d corrected	Table 4-15 on page 4-11
Purpose description extended	*Power Control Register* on page 4-41

Table C-5 Differences between issue D and issue F (continued)

Change	Location
Configurations description corrected	*Configuration Base Address Register* on page 4-42
Chapter renamed	Chapter 5 *Jazelle DBX registers*
6.1 application specific corrected to address space specific	*About the MMU* on page 6-2
Unified Main TLB description clarified	*Memory Management Unit* on page 6-2
Duplicate information about page sizes removed	
ASID description corrected and extended, and cross-reference added	
TLB match process duplicate information about page sizes removed	*TLB match process* on page 6-4
Synchronous and asynchronous aborts incorrect cross-reference removed	*Synchronous and asynchronous aborts* on page 6-8
Cache features cross-reference corrected	*Cache features* on page 7-2
Implementation information removed	
Return stack predictions ARM or Thumb state replaced by instruction state	*Return stack predictions* on page 7-7
DSB section added	*About DSB* on page 7-10
AXI master 0 interface attributes corrections to values	Table 8-1 on page 8-2
Debug chapter moved to before PMU chapter	
Figure redrawn	Figure 10-2 on page 10-3
Corrections to bit format	Table 10-1 on page 10-5
Footnote about **CLUSTERID** values added	Table 4-16 on page 4-12
Value column added	Table 10-10 on page 10-14
DBGCPUDONE description extended	*DBGCPUDONE* on page 10-18
PMU management registers section added	*PMU management registers* on page 11-5
Signal descriptions extended	*Configuration signals* on page A-5
Signal descriptions extended, information repeated from AXI removed	Table A-8 on page A-8
AWBURSTM0[1:0]	
AWLENM0[3:0]	
AWLOCKM0[1:0]	
Signal descriptions extended, information repeated from AXI removed	Table A-11 on page A-10
ARLENM0[3:0]	
ARLOCKM0[1:0]	
Title changed	*AXI Master1 signals instruction accesses* on page A-11
Information repeated from AXI removed	Table A-14 on page A-12

Table C-5 Differences between issue D and issue F (continued)

Change	Location
ARLENM1[3:0]	
PMUEVENT[46] and **PMUEVENT[47]** corrected	Table A-17 on page A-14
Introduction reduced, and note about `DSB` behavior added.	*Serializing instructions* on page B-9

Table C-6 Differences between issue F and issue G

Change	Location	Affects
Update description of transition from standby to run mode	*Standby modes* on page 2-11	All revisions
Addition of REVIDR -	*c15 registers* on page 4-11	r3p0
	Revision ID register on page 4-20	r3p0
Data cache no longer supports round robin replacement policy	Table 4-35 on page 4-25 *Memory system* on page 7-2	From r2p0
Update description of accessing the Jazelle Configurable Opcode Translation Table Register	*Jazelle Configurable Opcode Translation Table Register* on page 5-8	All revisions
Clarified implementation-defined aspect of invalidating TLBs	*About the L1 memory system* on page 7-2	All revisions
Added information about cache policies	*Cortex-A9 behavior for Normal Memory Cacheable memory regions* on page 7-8	All revisions
AWUSERM0[8:0] encodings table corrected	Table 8-5 on page 8-5	All revisions
Update the introduction to debug register features	*Debug register features* on page 10-4	All revisions
Remove reference to PMU registers from Debug chapter	*CP14 Debug register summary* on page 10-5	All revisions
Update introduction to debug register summary	*Debug register summary* on page 10-5	All revisions
Remove reference to DBGDSCCR	Table 10-1 on page 10-5 *Debug register descriptions* on page 10-7	All revisions
Update description of BVR	Table 10-5 on page 10-10	All revisions
Move debug management registers information from debug registers summary to debug management registers	Table 10-1 on page 10-5 Table 10-9 on page 10-13	All revisions
Update description of debug management registers	*Debug management registers* on page 10-13	All revisions
Update description of DBGITCTRL and DBGDEVID registers	Table 10-9 on page 10-13	All revisions
Update description of external debug interface	*External debug interface* on page 10-16	All revisions

Table C-6 Differences between issue F and issue G (continued)

Change	Location	Affects
Update introduction to PMU register summary	*PMU register summary* on page 11-3	All revisions
Remove reference to Processor ID Registers from Debug chapter	Table 11-1 on page 11-3	All revisions
Update descriptions of PMICTRL and PMDEVID	Table 11-2 on page 11-5	All revisions
Update description of PMU management registers	*PMU management registers* on page 11-5	All revisions
Update description of performance monitoring events	*Performance monitoring events* on page 11-7	All revisions
Updated description of **PENABLEDBG** signal	Table A-26 on page A-22	All revisions
CoreLink Level 2 Cache Controller renamed	Throughout document	All revisions