Cortex™-A9

Revision: r2p0

Technical Reference Manual

ARM®

ARM DDI 0388E (ID113009)

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

Proprietary Notice

Confidentiality Status

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

Unrestricted Access is an ARM internal classification.

Product Status

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

Web Address

`http://www.arm.com`

Contents
Cortex-A9 Technical Reference Manual

List of Tables
Cortex-A9 Technical Reference Manual

List of Figures
Cortex-A9 Technical Reference Manual

Preface

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

About this manual

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 manual

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.

Chapter 3 *Programmers Model*

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

Chapter 4 *The System Control Coprocessors*

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

Chapter 5 ***Memory Management Unit***

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

Chapter 6 ***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 7 ***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 8 ***Preload Engine***

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

Chapter 9 ***Performance Monitoring Unit***

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

Chapter 10 ***Debug***

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

Appendix A ***Signal Descriptions***

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

Appendix B ***Instruction Cycle Timings***

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.

Glossary Read this for definitions of terms used in this book.

Conventions

Conventions that this book can use are described in:

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

Typographical

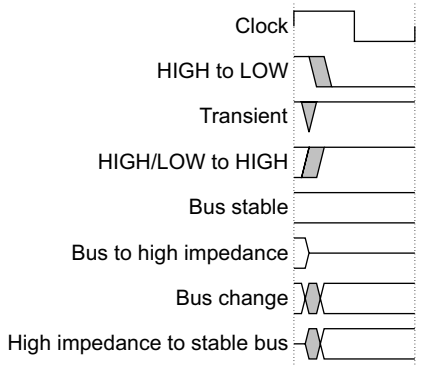
The typographical conventions are:

italic	Highlights important notes, introduces special terminology, denotes internal 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.
`mono`space	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>, <opc2>`

Timing diagrams

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

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



Key to timing diagram conventions

Signals

The signal conventions are:

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.

Prefix A Denotes *Advanced eXtensible Interface* (AXI) global and address channel signals.

Prefix AF Denotes *Advanced Trace Bus* (ATB) flush control signals.

Prefix AR Denotes AXI read address channel signals.

Prefix AT Denotes ATB data flow signals.

Prefix AW Denotes AXI write address channel signals.

Prefix B Denotes AXI write response channel signals.

Prefix C Denotes AXI low-power interface signals.

Prefix H Denotes *Advanced High-performance Bus* (AHB) signals.

Prefix P Denotes *Advanced Peripheral Bus* (APB) signals.

Prefix R Denotes AXI read channel signals.

Prefix W Denotes AXI write channel signals.

Additional reading

This section lists publications by ARM and by third parties.

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

ARM publications

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

- *ARM Architecture Reference Manual, ARMv7-A and ARMv7-R edition* (ARM DDI 0406)
- *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 TRM* (ARM DDI 0401)
- *CoreSight PTM-A9 Integration Manual* (ARM DII 0162)
- *CoreSight Program Flow Trace™ Architecture Specification*,v1.0 (ARM IHI 0035)
- *PrimeCell Level 2 Cache Controller (PL310) Technical Reference Manual* (ARM DDI 0246)
- *AMBA® AXI Protocol v1.0 Specification* (ARM IHI 0022)
- *ARM Generic Interrupt Controller Architecture Specification* (ARM IHI 0048)
- *RealView ICE User Guide* (ARM DUI 0155)
- *Intelligent Energy Controller Technical Overview* (ARM DTO 0005).
- *CoreSight Architecture Specification* (ARM IHI 0029).
- *CoreSight Technology System Design Guide* (ARM DGI 0012).

- *The ARM Cortex-A9 Processors* White paper.

Other publications

This section lists relevant documents published by third parties:

- *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 if appropriate.

Feedback on this book

If you have any comments on this book, send e-mail to `errata@arm.com`. Give:

- the title
- the number
- the relevant page number(s) 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 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 a PL310 L2 Cache Controller,

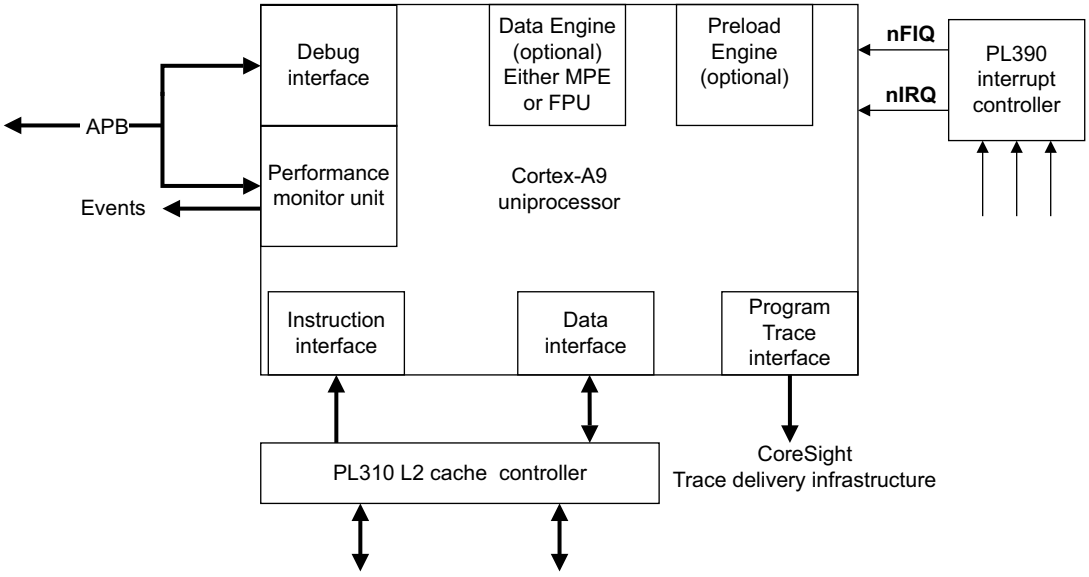


Figure 1-1 Cortex-A9 uniprocessor system.

1.1.1 Data Engine

The design can include a Data Engine. The following sections describe the Data Engine options:

- *Media Processing Engine* on page 1-3
- *Floating-Point Unit* on page 1-3.

Media Processing Engine

The optional *Media Processing Engine* (MPE) implements ARM NEON technology, a media and signal processing architecture that adds instructions targeted at audio, video, 3-D graphics, image, and speech processing. Advanced SIMD instructions are available in both ARM and Thumb states.

The optional MPE also implements a VFPv3-D32 Floating-Point Unit.

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. It can generate an Undefined instruction exception on vector instructions that lets the programmer emulate vector capability in software.

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

1.1.2 System design components

This section describes the PrimeCell components in Figure 1-1 on page 1-2 in the following sections:

- *PrimeCell Generic Interrupt Controller*
- *PrimeCell Level 2 Cache Controller (PL310)*.

PrimeCell Generic Interrupt Controller

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.

PrimeCell Level 2 Cache Controller (PL310)

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 PrimeCell 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
- 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* (AMBA) L2 interfaces.
- an *Accelerator Coherency Port* (ACP), 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 4-7
- *Auxiliary Control Register* on page 4-17
- *Configuration Base Address Register* on page 4-38.

Some PMU event signals have Cortex-A9 MPCore uses. See *Performance monitoring signals* on page A-17.

1.3 Compliance

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

- ARM Thumb®-2 32-bit instruction set architecture for overall code density comparable with Thumb and performance comparable with ARM instructions. See the *ARM Architecture Reference Manual* for details of both the ARM and Thumb instruction sets.
- *Thumb Execution Environment* (ThumbEE) to enable execution environment acceleration. See the *ARM Architecture Reference Manual* for details of the ThumbEE instruction set.
- Security Extensions architecture technology for enhanced security features. The Security Extensions architecture, its associated implementations, and supporting software, are commonly referred to as TrustZone. See *Security extensions overview* on page 3-8. See the *ARM Reference Manual* for details on how TrustZone works in the architecture.
- Advanced SIMD architecture extension to accelerate the performance of multimedia applications such as 3-D graphics and image processing. The Advanced SIMD architecture extension, its associated implementations, and supporting software, are commonly referred to as NEON technology. See the *ARM Architecture Reference Manual* for details of the NEON technology.

  See the *Cortex-A9 NEON Media Processing Engine Technical Reference Manual* for implementation-specific information.
- *Vector Floating-Point v3* (VFPv3) architecture for floating-point computation that is compliant with the IEEE 754 standard. See the *ARM Architecture Reference Manua*l for details of the VFPv3 subarchitecture.

  See the *Cortex-A9 Floating-Point Unit Technical Reference Manual* for implementation-specific information.
- The processor implements the ARMv7 Debug architecture that includes support for TrustZone and CoreSight. The Cortex-A9 processor implements Baseline CP14, Extended CP14 debug access, and memory mapped access to the debug registers. See Chapter 10 *Debug* for more information.

1.4 Features

The Cortex-A9 processor features are:

- superscalar, variable length, out-of-order pipeline with dynamic branch prediction
- ARM, Thumb, and ThumbEE instruction set support
- TrustZone security extensions
- Harvard level 1 memory system with:
  - *Memory Management Unit* (MMU)
- two 64-bit AXI master interfaces:
  - Master0 is the data side bus
  - Master1 is the instruction side bus. It has no write channel.
- v7 debug architecture
- trace support
  - *Program Trace Macrocell* (PTM) interface
- *Intelligent Energy Manager* (IEM) support with
  - three voltage domains.
- optional Preload Engine
- optional Jazelle hardware acceleration
- optional Data Engine:
  - Media Processing Engine

    The MPE has NEON technology and VFPv3-D32 FPU with trapless execution.
  - VFPv3-D16 FPU with trapless execution.

1.5 Interfaces

The processor has the following external interfaces:

- AMBA AXI interfaces
- v7 compliant debug interface, including an APBv3 external debug interface
- *Design for Test* (DFT) interface.

See the *AMBA AXI Protocol Specification*, the *CoreSight Architecture Specification*, and the *Cortex-A9 MBIST Controller Technical Reference Manual* for more information on these interfaces.

1.6 Configurable options for the Cortex-A9 processor

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

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.

For details, see the *Cortex-A9 MBIST Controller Technical Reference Manual*.

1.7 Test features

There are no test features for the Cortex-A9 processor.

1.8 Product documentation, design flow, and architecture

This section describes the Cortex-A9 family books, how they relate to the design flow, and the relevant architectural standards and protocols.

See *Additional reading* on page xx for more information about the books described in this section.

1.8.1 Documentation

The Cortex-A9 family 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. It is required at all stages of the design flow. Some behavior described in the TRM might not be relevant because of the way that the Cortex-A9 processor is implemented and integrated.

- 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 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
- the integrator to determine the pin configuration of the SoC that you are using.

Configuration and Sign-Off Guide

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

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

The ARM product deliverables include reference scripts and information about using them to implement your design. Reference methodology 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 processor is delivered as synthesizable RTL. Before it can be used in a product, it must go through the following process:

1. Implementation. The implementer configures and synthesizes the RTL to produce a hard macrocell. If appropriate, this includes integrating the RAMs into the design.
2. Integration. The integrator connects the implemented design into a SoC, This includes connecting it to a memory system and peripherals.
3. Programming. The system programmer develops the software required to configure and initialize the processor, and tests the required application software.

Each stage of the process:

- can be performed by a different party
- can include options that affect the behavior and features at the next stage:

  Build configuration

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

  Configuration inputs

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

  Software configuration

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

Note

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

1.8.3 Architecture and protocol information

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

- *ARM architecture*
- *Trace macrocell, optional*
- *Advanced Microcontroller Bus Architecture*.

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

ARM architecture

The Cortex-A9 processor implements the ARMv7-A architecture profile. See the *ARM Architecture Reference Manual, ARMv7-A and ARMv7-R edition*.

Trace macrocell, optional

The Cortex-A9 processor implements the v1.0 PFT architecture. See the *CoreSight Program Flow Trace Architecture Specification, v1.0*.

Advanced Microcontroller Bus Architecture

This Cortex-A9 processor complies with the AMBA 3 protocol. See the *AMBA AXI Protocol v1.0 Specification* and the *AMBA 3 APB Protocol Specification*.

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*.

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 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 4-36
- 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-17.
- TLB options for 128 entries or 64 entries. See *TLB Type Register* on page 4-6.
- **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 4-13.
- Addition of PL310 cache controller optimization description. See *Optimized accesses to the L2 memory interface* on page 7-7.
- Change to the serializing behavior of `DMB`. See *Serializing instructions* on page B-10.
- ID Register values changed to reflect r1p0 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 4-23.
  - Preload Engine registers added. See *c11 system control registers summary table* on page 4-33.
  - Preload operations added and MCRR instruction added. See Chapter 8 *Preload Engine*.
  - Addition of Preload Engine events.

    See *Performance monitoring* on page 2-3, Table 9-2 on page 9-4, and Table A-18 on page A-17.
- Change to voltage domains. See Figure 2-4 on page 2-17
- NEON busy register. See *NEON busy Register* on page 4-37
- ID Register values changed to reflect r2p0 revision.

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.

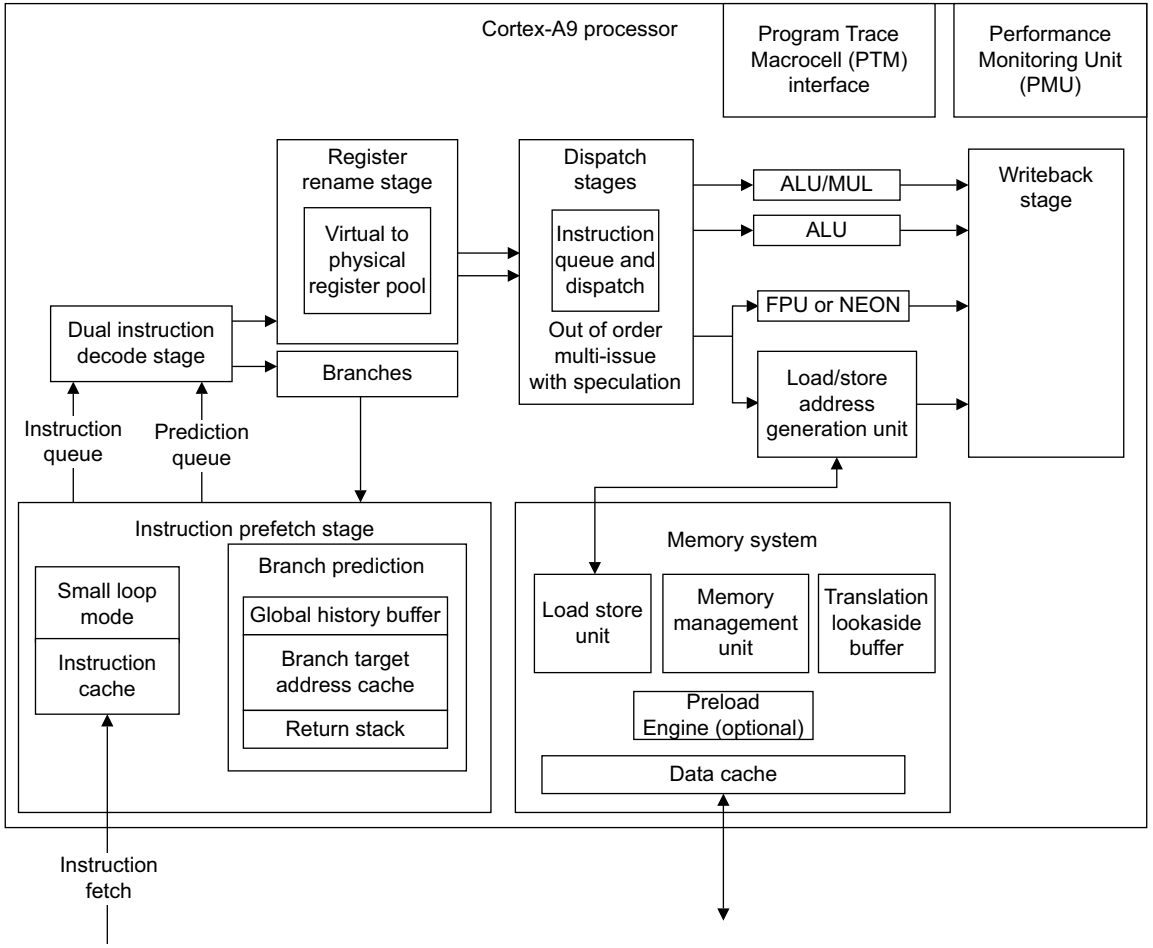


Figure 2-1 Cortex-A9 processor top-level diagram

2.1.1 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.

2.1.2 Small loop mode

Small loop mode provides low power operation while executing small instruction loops. See *Energy efficiency features* on page 2-12.

2.1.3 PTM interface

The Cortex-A9 processor optionally implements a *Program Trace Macrocell* (PTM) interface, which 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 the Program Trace Macrocell interface* on page 2-5.

2.1.4 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 9 *Performance Monitoring Unit*

2.1.5 Virtualization of interrupts

With virtualized interrupts a guest *Operating System* (OS) can use a modified version of the exception behavior model to speed up handling of interrupts

See *Virtualization Control Register* on page 4-25.

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 4-23.

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:

- AMBA AXI interfaces
- APB CoreSight interface
- DFT interface.

See the *AMBA AXI Protocol Specification*, the *CoreSight Architecture Specification*, the *CoreSight PFT Architecture Specification*, and the *Cortex-A9 MBIST Controller Technical Reference Manual* for more information on these interfaces.

2.2.1 Program Flow Trace and the Program Trace Macrocell interface

In addition, the Cortex-A9 processor implements the *Program Flow Trace* (PFT) architecture protocol. The following sections describe the Cortex-A9 *Program Trace Macrocell* (PTM) interface:

- *About the PTM interface*
- *Prohibited regions*.

About the PTM interface

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 Cortex-A9 PFT Architecture Specification* and the *CoreSight Cortex-A9 PTM Technical Reference Manual* for more information about tracing with waypoints.

Prohibited regions

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.

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.

Figure 2-2 on page 2-6 shows the PTM interface signals.

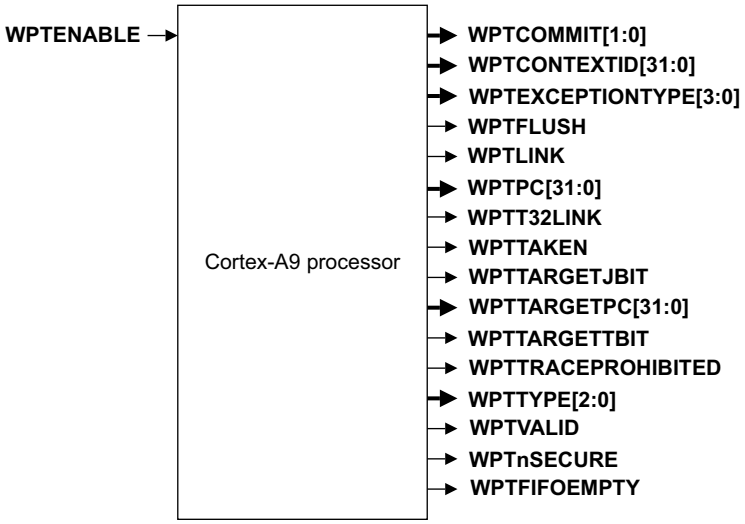


Figure 2-2 PTM interface signals

See Appendix A *Signal Descriptions* and the *CS Cortex-A9 Program Trace Macrocell TRM* for more information.

2.3 Clocking

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

2.3.1 Synchronous clocking

The Cortex-A9 processor 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**.

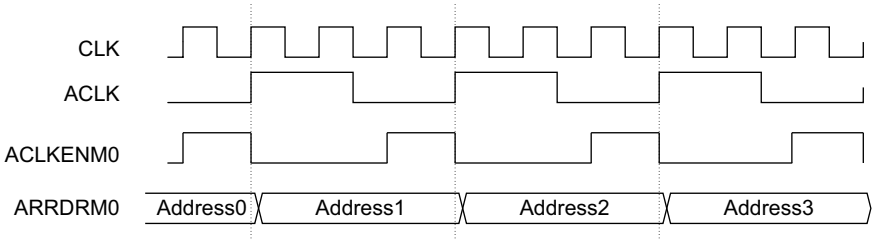


Figure 2-3 ACLKENM0 used with a 3:1 clock ratio

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

2.4 Dynamic high level clock gating

Dynamic high level clock gating is described in the following sections:

- *Gated blocks*
- *Power Control Register*
- *Effects of max_clk latency bits*
- *Dynamic high level clock gating activity* on page 2-9.

2.4.1 Gated blocks

The Cortex-A9 processor or each processor in a CortexA9MP Core design supports dynamic high level clock gating of:

- the integer core
- the system control block.
- the Data Engine, if implemented.

2.4.2 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 4-36.

2.4.3 Effects of max_clk latency bits

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 it clock cut receives active signals on the first clock edge of the wake-up. This gives optimum performance.

2.4.4 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.

2.5 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.

2.5.1 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
Debug logic reset	1	1	0
No reset, normal run mode	1	1	1

2.5.2 Power-on reset

You must apply power-on or *cold* reset to the Cortex-A9 processor 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.

On power-on, perform the following reset sequence:

1. Apply all resets.
2. Apply at least 9 **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 for example applying 15 cycles on every clock domain.
3. Stop the **CLK** clock. 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 resets.
6. Wait for the equivalent of another approximately 10 cycles, again to compensate for clock and reset tree latencies.
7. Restart the clock.

2.5.3 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.

2.5.4 MPE SIMD logic reset

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 to that the power to the SIMD part of the MPE can be safely switched on or off. See Table 2-2 on page 2-13.

2.5.5 Debug reset

Use **nDBGRESET** to reset the debug hardware within the Cortex-A9 processor, including breakpoints and watchpoints values.

2.6 Power management

The processor provides mechanisms to control both dynamic and static power dissipation. Static power control is implementation-specific. The following sections describe:

- Energy efficiency features
- Cortex-A9 processor power control.

2.6.1 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 look-ups 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
- small loop mode.Instruction loops that are smaller than 64 bytes often complete without additional instruction cache accesses, so lowering power consumption.

In the Cortex-A9 processor, extensive use is also made of gated clocks and gates to disable inputs to unused functional blocks. Only the logic in use to perform an operation consumes any dynamic power.

2.6.2 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	Comments
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 4-19 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		
WFI/WFE	Powered-up	Powered-up	Powered Up	WFI/WFE mode, see *Wait for interrupt (WFI/WFE) mode* on page 2-14.
		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.

Wait for interrupt (WFI/WFE) mode

Wait for Interrupt mode disables most of the clocks of 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 from the WFI state.

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

- an interrupt, masked or unmasked
- an asynchronous data abort, regardless of the value of the CPSR.A bit. A pending wake-up event prevents the processor from entering low power mode.
- a debug request, regardless of whether debug is enabled
- a reset.

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

- an interrupt, unless masked
- a debug request, regardless of whether debug is enabled
- a previous exception return on the same processor
- a reset
- the assertion of the **EVENTI** input signal.

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.

Entry into WFI Mode is performed by executing the `WFI` Wait For Interrupt instruction.

Entry into WFE Mode is performed by executing the `WFE` Wait For Event instruction.

To ensure that the memory system is not affected by the entry into the WFI state, perform a Data Synchronization Barrier, to ensure that all explicit memory accesses occurring in program order before the WFI/WFE complete. This avoids any possible deadlocks that can be caused in a system where memory access can trigger or enable an interrupt that the Cortex-A9 processor is waiting for.

Any other memory accesses that have been started at the time that the `WFI` or `WFE` instruction is executed complete as normal. This ensures that the L2 memory system does not see any disruption caused by the `WFI`.

The debug channel remains active throughout a `WFI`.

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. All inputs to the RAMs must be clamped to a known logic level, with the chip enable held inactive. This clamping is not implemented in gates as part of the default synthesis flow because it can contribute to a tight critical path. Implementations that include Dormant mode must add these clamps around the RAMs, either as explicit gates in the RAM power domain, or as pull-down transistors that clamp the values while the Cortex-A9 processor is powered down.

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 2-16 for more information.
- Before removing the power, the Reset signal 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 *Standby and Wait For Event 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-28.

2.6.3 IEM Support

The IEM infrastructure is intended to be supported at the system level to enable you to choose at which level in the SoC to separate different voltage domains.

Placeholders between Cortex-A9 logic and RAM arrays are available so that implementation of level shifters for these parts can be in a different voltage domains.

2.6.4 Cortex-A9 voltage domains

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

- a voltage domain for Cortex-A9 processor logic cells
- a voltage domain for Cortex-A9 processor data engines
- a voltage domain for Cortex-A9 processor RAMs.

Figure 2-4 shows the voltage domains.

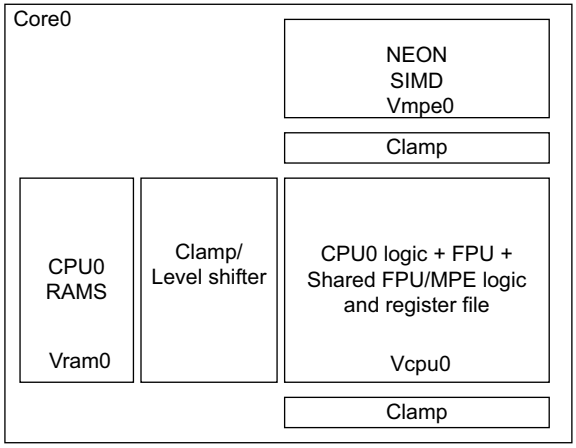


Figure 2-4 Voltage domains for Cortex-A9 r2p0 designs

The FPU is part of the CPU power domain. The FPU clock is based on the CPU 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.

2.7 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 ARMv7-A Architecture Reference Manual provides a complete description of the programmers model

3.2 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 *CP14 Jazelle registers* on page 4-43.

3.3 NEON technology

NEON technology is a media and signal processing architecture that adds instructions targeted primarily at audio, video, 3-D graphics, image, and speech processing. Advanced *Single Instruction Multiple Data* (SIMD) instructions are available in both ARM and Thumb states.

NEON technology includes both Advanced *Single Instruction Multiple Data* (SIMD) instructions and the ARM VFPv3 instructions.

See the *ARM Architecture Reference Manual* for details of the NEON technology.

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

3.4 Memory formats

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.

3.5 Addresses in the Cortex-A9 processor

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

When the Cortex-A9 processor is executing in Non-secure state, the processor performs translation table look-ups 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 look-ups 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 3-1 shows the address types in the processor system.

Table 3-1 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 look-up. 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 core 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 look-ups are secure even if the L1 entry is marked Non-secure.

3.6 Security extensions overview

The purpose of the security extensions is to enable the construction of a secure software environment. This section describes the following:

- *System boot sequence*

See the *ARM Architecture Reference Manual* for details of the security extensions.

3.6.1 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 make sure 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 further 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.

Chapter 4
The System Control Coprocessors

This chapter describes the purpose of the system control coprocessor, its structure, operation, and how to use it. It contains the following sections:

4.1 About the system control coprocessor

The purpose of the system control coprocessor is to control and provide 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.

4.2 Summary of system control coprocessor registers

This section shows summary tables of the register allocation and reset values of the system control coprocessor where:

- CRn is the register number within CP15
- Op1 is the Opcode_1 value for the register
- CRm is the operational register
- Op2 is the Opcode_2 value for the register.
- Type is:
  - Read-only (RO)
  - Write-only (WO)
  - Read/write (RW).
- Reset is the reset value of the register.

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 8-11. Reserved register addresses are RAZ/WI.

This section does not reproduce information about registers already described in the *ARM Architecture Reference Manual*. This chapter describes the implementation-defined control coprocessor registers.

4.2.1 Deprecated registers

In ARMv7A 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 Architecture v7, and are not implemented in the Cortex-A9 processor.

4.2.2 System control registers

This following section describe the system control registers:

- *c0 summary table* on page 4-5
- *TLB Type Register* on page 4-6
- *Multiprocessor Affinity Register* on page 4-7
- *Cache Size Identification Register* on page 4-8

4.2.3 c0 summary table

Table 4-1 shows the system control registers when CRn is c0.

Table 4-1 c0 system control registers

Op1	CRm	Op2	Name	Type	Reset	Description
0	c0	0	MIDR	RO	0x411FC090	Main ID Register
		1	CTR	RO	0x83338003	Cache Type Register
		2	TCMTR	RO	0x00000000	TCM Type Register
		3	TLBTR[a]	RO	-	*TLB Type Register* on page 4-6
		5	MPIDR	RO	-	*Multiprocessor Affinity Register* on page 4-7
	c1	0	ID_PFR0	RO	0x00001231	Processor Feature Register 0
		1	ID_PFR1	RO	0x00000011	Processor Feature Register 1
		2	ID_DFR0	RO	0x00010444	Processor Feature Register 2
		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 4-8
		1	CLIDR	RO	0x09000003	*Cache Level ID Register* on page 4-10
		7	AIDR	RO	0x00000000	*Auxiliary ID Register* on page 4-11
2	c0	0	CSSELR	RW	-	*Cache Size Selection Register* on page 4-12

a. Depends on TLBSIZE. See *TLB Type Register* on page 4-6.

4.2.4 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 4-1 on page 4-5.

Figure 4-1 shows the TLBTR bit assignments.

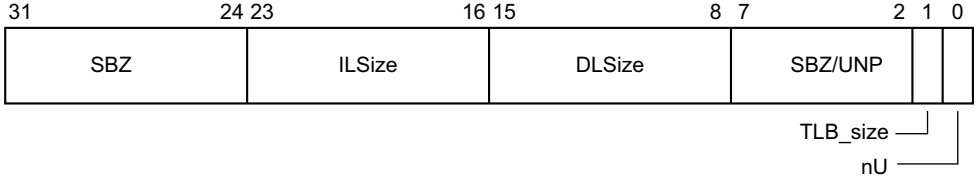


Figure 4-1 TLBTR bit assignments

Table 4-2 shows the TLBTR bit assignments.

Table 4-2 TLBTR bit assignments

Bits	Name	Description
[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	1 the TLB has 128 entries, 0 the TLB has 64 entries
[0]	nU	Specifies if the TLB is unified, 0, or if there are separate instruction and data TLBs, 1. For the Cortex-A9 processor this is 0.

To access the TLBTR, use:

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

4.2.5 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 4-1 on page 4-5.

Figure 4-2 shows the MPIDR bit assignments.

Figure 4-2 MPIDR bit assignments

Table 4-3 shows the MPIDR bit assignments.

Table 4-3 MPIDR bit assignments

Bits	Name	Description
[31]	-	Indicates the register uses the new multiprocessor format. This is always 1.
[30]	U bit	0 = Processor is part of a multiprocessor system. 1 = Processor is part of a uniprocessor system.
[29:12]	-	SBZ.
[11:8]	Cluster ID	Value read in **CLUSTERID** configuration pins. It identifies a Cortex-A9 MPCore processor in a system with more than one Cortex-A9 MPCore processor present. SBZ in a uniprocessor configuration.
[7:2]	-	SBZ.
[1:0]	CPU ID	The value depends on the number of configured processors. • One Cortex-A9 processor, the CPU ID is `0x0`. • Two Cortex-A9 processors, the CPU IDs are `0x0 and 0x1`. • Three Cortex-A9 processors, the CPU IDs are `0x0`, `0x1`, and `0x2`. • Four Cortex-A9 processors, the CPU IDs are 0x0, `0x1`, `0x2`,and `0x3`.

To access the MPIDR, use:

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

4.2.6 Cache Size Identification Register

The CCSIDR characteristics are:

Purpose Provides information about the architecture of the caches

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 4-7 on page 4-13.

Figure 4-3 shows the CCSIDR bit assignments.

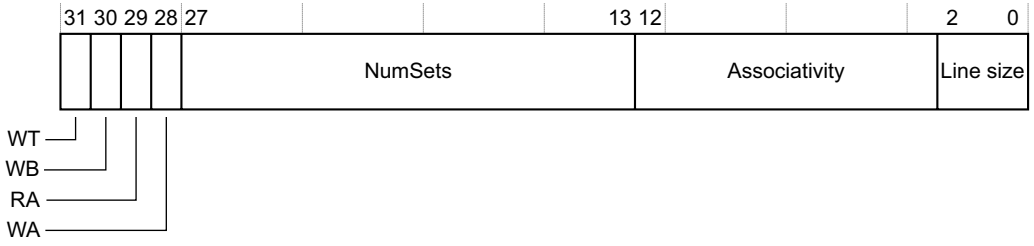


Figure 4-3 CCSIDR bit assignments

Table 4-4 shows how the CSSIDR bit assignments.

Table 4-4 CCSIDR bit assignments

Bits	Name	Description
[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.

Table 4-4 CCSIDR bit assignments (continued)

Bits	Name	Description
[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, use:

```
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 4-12.

4.2.7 Cache Level ID Register

The CLIDR characteristics are:

Purpose Indicates the cache levels that are implemented.

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 4-7 on page 4-13.

Figure 4-4 shows the CLIDR bit assignments.

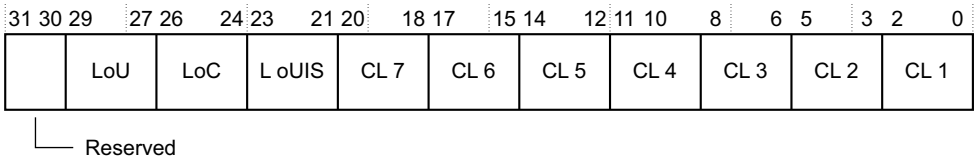


Figure 4-4 CLIDR bit assignments

Table 4-5 shows the CLIDR bit assignments.

Table 4-5 CLIDR bit assignments

Bits	Name	Description
[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, use:

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

4.2.8 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 4-7 on page 4-13.

To access the Auxiliary Level ID Register, use:

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

Note

The AIDR is unused in this implementation.

4.2.9 Cache Size Selection Register

The CSSELR characteristics are:

Purpose Selects the current CCSIDR.

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 4-1 on page 4-5.

Figure 4-5 shows the CSSELR bit assignments.

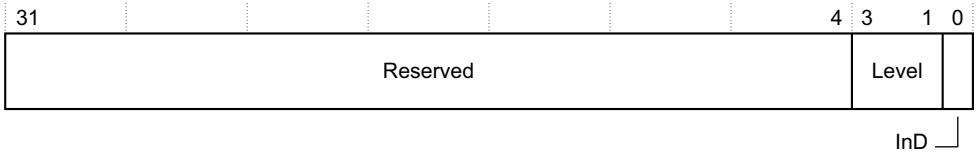


Figure 4-5 CSSELR bit assignments

Table 4-6 shows the CSSELR bit assignments.

Table 4-6 CSSELR bit assignments

Bits	Name	Description
[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	1 = Instruction cache 0 = Data cache.

To access the CSSELR, use:

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

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

4.2.10 c1 summary table

Table 4-7 shows the system control registers when CRn is c1.

Table 4-7 c1 system control registers

Op1	CRm	Op2	Name	Type	Reset	Description
0	c0	0	SCTLR	RW	0x00C50078	*System Control Register*
		1	ACTLR[a]	RW	0x00000000	*Auxiliary Control Register* on page 4-17
		2	CPACR	RW	[b]	*Coprocessor Access Control Register* on page 4-19
	c1	0	SCR[c]	RW	0x00000000	Secure Configuration Register[d]
		1	SDER[c]	RW	0x00000000	*Secure Debug Enable Register* on page 4-22
		2	NSACR	RW[e]	[f]	*Non-secure Access Control Register* on page 4-23
		3	VCR[c]	RW	0x00000000	*Virtualization Control Register* on page 4-25

a. RO in non secure state if NSACR[18]=0 and RW if NSACR[18]=1.
b. 0x00000000 if NEON present and 0xC0000000 if NEON not present.
c. No access in Non-secure state.
d. Early termination is permitted. That is, execution time of data operations can depend on the data values.
e. This is a read and write register in Secure state and a read-only register in the Non-secure state.
f. 0x00000000 if NEON present and 0x0000C000 if NEON not present.

4.2.11 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. *System Control Register* on page 4-13 shows banked and secure modify only bits.

Configurations Available in all configurations.

Attributes See the register summary in Table 4-7 on page 4-13.

Figure 4-6 shows the SCTLR bit assignments.

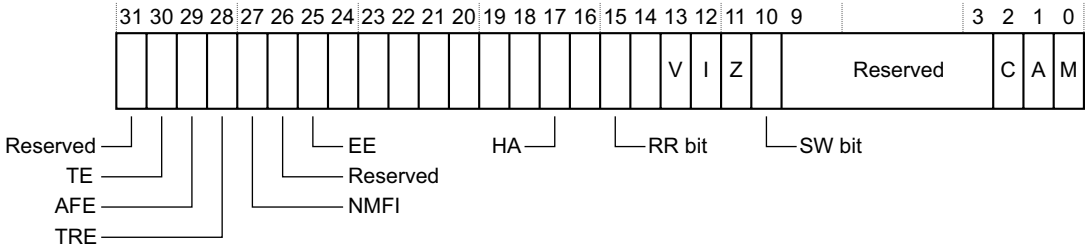


Figure 4-6 SCTLR bit assignments

Table 4-8 shows the SCTLR bit assignments.

Table 4-8 SCTLR bit assignments

Bits	Name	Access	Description
[31]	-	-	SBZ.
[30]	TE	Banked	TE, 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	This is the 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	NMFI, nonmaskable The reset value is determined by **CFGNMFI**. The bit cannot be configured by software. This bit is read-only.

Table 4-8 SCTLR bit assignments (continued)

Bits	Name	Access	Description
[26]	-	-	RAZ/SBZP
[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. **CFGEND** sets the reset value. 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 look-ups. 0 = little-endian 1 = big-endian.
[24]	-	-	RAZ/WI
[23:22]	-	-	RAO/SBOP
[21]	-	-	RAZ/WI
[20:19]	-	-	RAZ/SBZP
[18]	-	-	RAO/SBOP
[17]	HA	-	RAZ/WI Hardware management access flag disabled.
[16]	-	-	RAO/SBOP
[15]	-	-	RAZ/SBZP
[14]	RR	Secure modify only	Replacement strategy for caches, BTAC, and micro TLBs. This bit is R/W 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 re-mapped. 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.

Table 4-8 SCTLR bit assignments (continued)

Bits	Name	Access	Description
[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.
[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, use:

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

4.2.12 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.
- PL310 cache allocation:
  - write full line of zeros mode.

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 the other Cortex-A9 processors in the same coherent cluster.

Attributes See the register summary in Table 4-7 on page 4-13.

Figure 4-7 shows the ACTLR bit assignments.

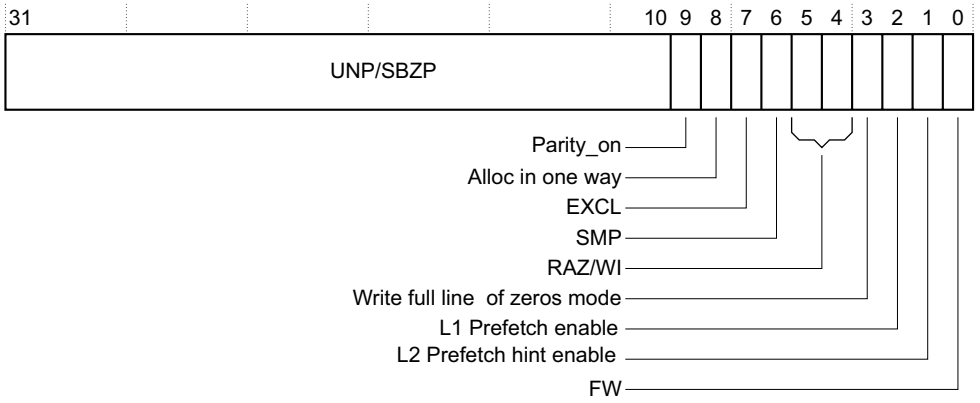


Figure 4-7 ACTLR bit assignments

Table 4-9 shows the ACTLR bit assignments.

Table 4-9 ACTLR bit assignments

Bits	Name	Description
[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 memcpy operations to reduce cache pollution. The reset value is zero.
[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.

Table 4-9 ACTLR bit assignments (continued)

Bits	Name	Description
[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 present.

a. This feature must be enabled only when the slaves connected on the Cortex-A9 AXI master port support it. The PL310 Cache Controller supports this feature. See *Optimized accesses to the L2 memory interface* on page 7-7.

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

```
MRC p15, 0,<Rd>, c1, c0, 1; Read ACTLRs
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.

4.2.13 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, the debug control coprocessor, or CP15, the system control coprocessor.

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 4-7 on page 4-13.

Figure 4-8 shows the CPACR bit assignments.

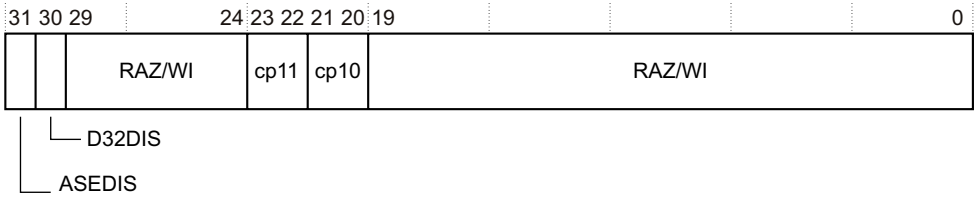


Figure 4-8 CPACR bit assignments

Table 4-10 shows the CPACR bit assignments.

Table 4-10 CPACR bit assignments

Bits	Name	Description
[31]	ASEDIS	Disable Advanced SIMD Extension functionality 0 = This bit does not cause any instructions to be undefined. 1 = All instruction encodings identified in the *ARM Architecture Reference Manual* as being part of the Advanced SIMD Extensions but that are not VFPv3 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, no NEON, RAO/WI. If implemented without both VFP and NEON, UNK/SBZP.
[30]	D32DIS	Disable use of D16-D31 of the VFP register file 0 = This bit does not cause any instructions to be undefined. 1 = All instruction encodings identified in the *ARM Architecture Reference Manual* as being VFPv3 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, no NEON, RAO/WI.If implemented without both VFP and NEON, UNK/SBZP.
[29:24]	-	RAZ/WI.

Table 4-10 CPACR bit assignments (continued)

Bits	Name	Description
[23:22]	cp11	Defines access permissions for the coprocessor. Access denied is the reset condition and is the behavior for nonexistent 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 nonexistent 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 4-23.

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 4-23.

To access the CPACR, use:

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

You must execute an `ISB` immediately after an update of the CPACR. See Memory Barriers in the *ARM Architecture Reference Manual*. 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.

4.2.14 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 4-7 on page 4-13.

Figure 4-9 shows the SDER bit assignments.

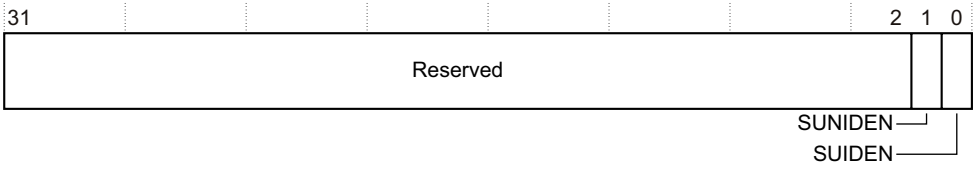


Figure 4-9 SDER bit assignments

Table 4-11 shows the SDER bit assignments.

Table 4-11 SDER bit assignments

Bits	Name	Description
[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, use:

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

4.2.15 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 and 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 4-7 on page 4-13.

Figure 4-10 shows the NSACR bit assignments.

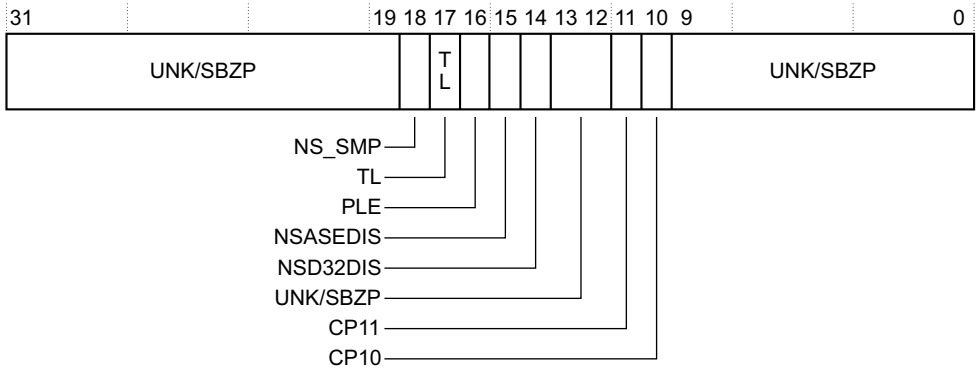


Figure 4-10 NSACR bit assignments

Table 4-12 shows the NSACR bit assignments.

Table 4-12 NSACR bit assignments

Bits	Name	Description
[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 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 UNDEF. 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 8 *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.

Table 4-12 NSACR bit assignments (continued)

Bits	Name	Description
[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, use:

```
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.

4.2.16 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 4-26 on page 4-35.

Figure 4-11 on page 4-26 shows the VCR bit assignments.

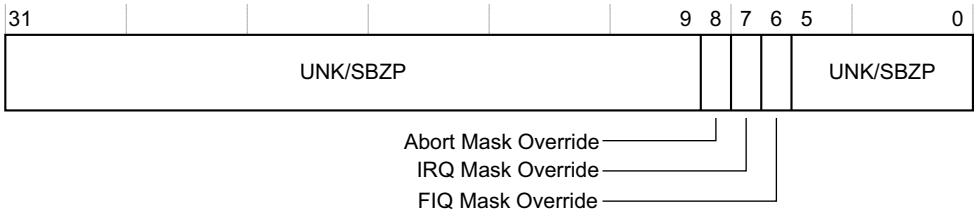


Figure 4-11 VCR bit assignments

Table 4-13 shows the VCR bit assignments.

Table 4-13 VCR bit assignments

Bits	Name	Description
[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, use:

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

4.2.17 c2 summary table

Table 4-14 shows the system control registers when CRn is c2.

Table 4-14 c2 system control registers

Op1	CRm	Op2	Name	Type	Reset	Description
0	c0	0	TTBR0	RW	-	Translation Table Base Register 0
		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.

4.2.18 c3 summary table

Table 4-15 shows the system control register when CRn is c3.

Table 4-15 c3 system control register

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

4.2.19 c4, c5, and c6 summary tables

There are no operations where CRn is c4.

Table 4-16 shows the system control registers when CRn is c5.

Table 4-16 c5 system control registers

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

Table 4-17 shows the system control registers when CRn is c6.

Table 4-17 c6 system control registers

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

4.2.20 c7 summary table

Table 4-18 shows the system control registers when CRn is c7.

Table 4-18 c7 system control registers

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 4-3
		6	BPIALL	WO	-	Cache operations registers
	c6	1	DCIMVAC	WO	-	
		2	DCISW	WO	-	
	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 4-3
		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 also *Deprecated registers* on page 4-3.

4.2.21 c8 summary table

Table 4-19 shows the system control registers when CRn is c8.

Table 4-19 c8 system control registers

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 also *Invalidate TLB Entries on ASID Match* on page 4-42.

4.2.22 c9 summary table

Table 4-20 shows the system control registers when CRn is c9.

Table 4-20 c9 system control registers

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 Selection Register
		2	PMXEVCNTR	RW	-	Performance Monitor 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 9 *Performance Monitoring Unit*.

4.2.23 c10 summary table

Table 4-21 shows the system control registers when CRn is c10.

Table 4-21 c10 system control registers

Op1	CRm	Op2	Name	Type	Reset	Description
0	c0	0	TLB Lockdown Register[a]	RW	0x00000000	*TLB Lockdown Register* on page 4-32
	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.

4.2.24 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 4-21 on page 4-31.

Figure 4-12 shows the TLB Lockdown Register bit assignments.

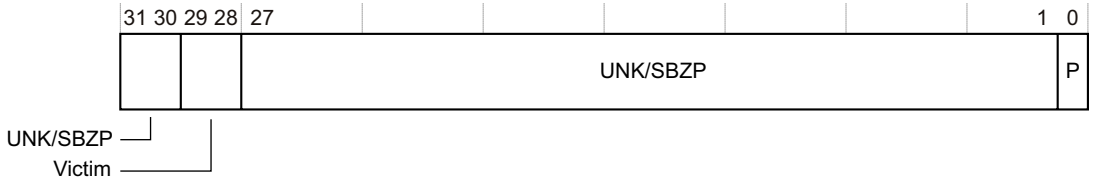


Figure 4-12 TLB Lockdown Register bit assignments

Table 4-22 shows the TLB Lockdown Register bit assignments

Table 4-22 TLB Lockdown Register bit assignments

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

To access the TLB Lockdown Register use:

```
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.

4.2.25 c11 system control registers summary table

Table 4-23 shows the system control registers where CRn is c11.

Table 4-23 c11 system control registers

Op1	CRm	Op2	Name	Type	Reset	Description
0	c0	0	PLEIDR	RO[a]	-	*PLE ID Register* on page 8-4
		2	PLEASR	RO[a]	-	*PLE Activity Status Register* on page 8-5
		4	PLEFSR	RO[a]	-	*PLE FIFO Status Register* on page 8-6
	c1	0	PLEUAR	Privileged R/W User RO	-	*Preload Engine User Accessibility Register* on page 8-7
		1	PLEPCR	Privileged R/W User RO	-	*Preload Engine Parameters Control Register* on page 8-8

a. RAZ if the PLE is not present.

See Chapter 8 *Preload Engine*.

4.2.26 c12 summary table

Table 4-24 shows the system control registers when CRn is c12.

Table 4-24 c12 system control registers

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	Interrupt Status Register	RO	0x00000000	Interrupt Status Register
		1	Virtualization Interrupt Register	RW[b]	0x00000000	*Virtualization Interrupt Register*

a. Only the secure version is reset to 0. The Non-secure version must be programmed by software
b. There is no access in Non-secure state.

4.2.27 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 4-24.

The virtual interrupt is delivered as soon as the processor is in NS state. Figure 4-13 shows the VIR bit assignments.

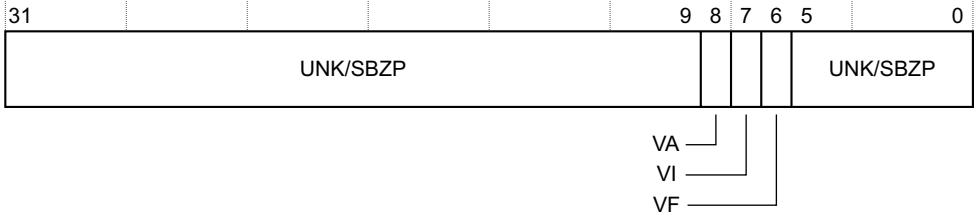


Figure 4-13 VIR bit assignments

Table 4-25 shows the Virtualization Interrupt Register bit assignments.

Table 4-25 Virtualization Interrupt Register bit assignments

Bits	Name	Description
[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, use:

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

4.2.28 c13 summary table

Table 4-26 shows the system control registers when CRn is c13.

Table 4-26 c13 system control registers

Op1	CRm	Op2	Name	Type	Reset	Description
0	c0	0	-	RW	0x00000000	*Deprecated registers* on page 4-3
		1	Context ID	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

4.2.29 c15 summary table

Table 4-27 shows the system control registers when CRn is c15.

Table 4-27 c15 system control registers

Op1	CRm	Op2	Name	Type	Reset	Description
0	c0	0	Power Control Register	RW[a]	[b]	*Power Control Register*
	c1	0	NEON busy Register	RO	0x00000000	*NEON busy Register* on page 4-37
4	c0	0	Configuration Base Address	RO[c]	[d]	*Configuration Base Address Register* on page 4-38
5	c4	2	Select Lockdown TLB Entry for read	WO[e]	-	on page 4-39*c15, TLB lockdown operations* on page 4-39
		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 Snoop Control Unit registers.
e. No access in Non-secure state.

4.2.30 Power Control Register

The Power Control Register characteristics are:

Purpose Enables you to set:
- the clock latency
- dynamic clock gating.

Usage constraints
- a read and 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 4-27.

Figure 4-14 shows the Power Control Register bit assignments.

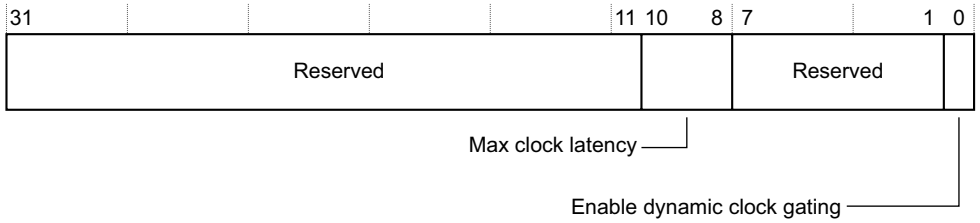


Figure 4-14 Power Control Register bit assignments

Table 4-28 shows the Power Control Register bit assignments.

Table 4-28 Power Control Register bit assignments

Bits	Name	Description
[31:11]	-	Reserved.
[10:8]	Max clock latency	Samples the value present on the **MAXCLKLATENCY** pins on exit from reset. This value reflects an implementation specific parameter, and ARM recommends that the software does not modify it.
[7:1]	-	Reserved.
[0]	Enable dynamic clock gating	Disabled at reset.

To access the Power Control Register, use:

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

4.2.31 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 4-27 on page 4-36.

Figure 4-15 on page 4-38 shows the NEON busy register bit assignments

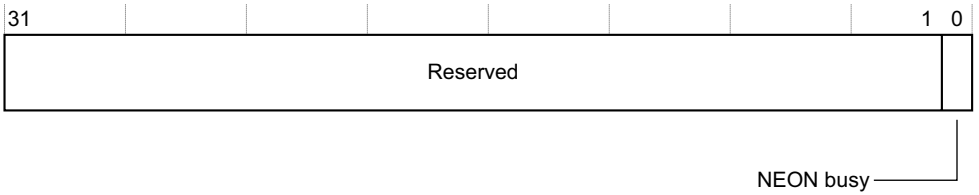


Figure 4-15 NEON busy register bit assignments

Table 4-29 shows the NEON busy Register bit assignments.

Table 4-29 Neon busy Register bit assignments

Bits	Name	Description
[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 NEON pipeline, or in the core pipeline

To access the NEON busy Register, use:

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

4.2.32 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 and 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 it is reset to **PERIPHBASE[31:13]** so that software can determine the location of the SCU registers.

Attributes See the register summary in Table 4-27 on page 4-36.

Figure 4-16 on page 4-39 shows the Configuration Base Address Register bit assignments.

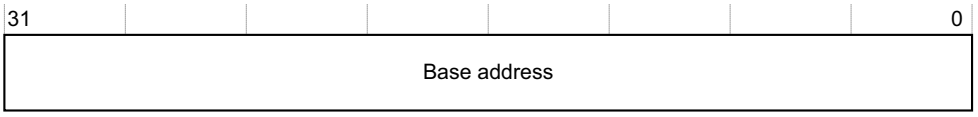


Figure 4-16 Configuration Base Address Register bit assignments

To access the Configuration Base Address Register, use:

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

4.2.33 c15, TLB lockdown operations

TLB lockdown operations enable saving or restoring lockdown entries in the TLB. Table 4-30 shows the defined TLB lockdown operations.

Table 4-30 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/read register when accessing TLB lockdown registers. Figure 4-17 on page 4-40 shows the bit assignment of the index register used to access the lockdown TLB entries.

Figure 4-17 Lockdown TLB index bit assignments

Figure 4-18 shows the bit arrangement of the TLB VA Register format.

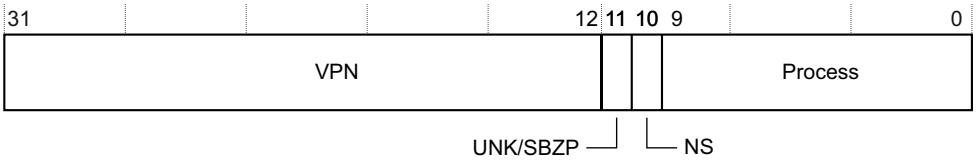


Figure 4-18 TLB VA Register bit assignments

Table 4-31 the TLB VA Register bit assignments.

Table 4-31 TLB VA Register bit assignments

Bits	Name	Description
[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 4-19 shows the bit arrangement of the memory space identifier.

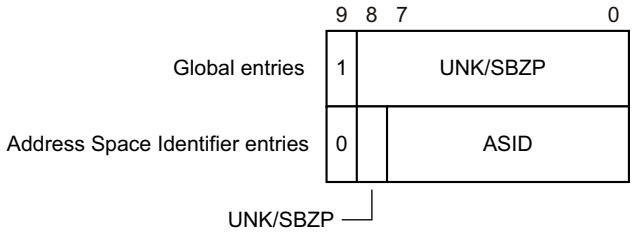


Figure 4-19 Memory space identifier format

Figure 4-20 on page 4-41 shows the TLB PA Register bit assignment.

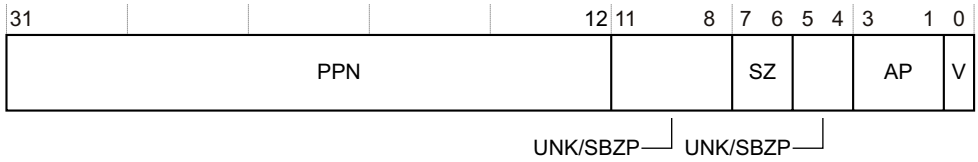


Figure 4-20 TLB PA Register bit assignments

Table 4-32 describes the functions of the TLB PA Register bits.

Table 4-32 TLB PA Register bit assignments

Bits	Name	Description
[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. See *Page sizes* on page 5-3.
[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 and 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 4-21 on page 4-42 shows the bit assignments of the TLB Attributes Register.

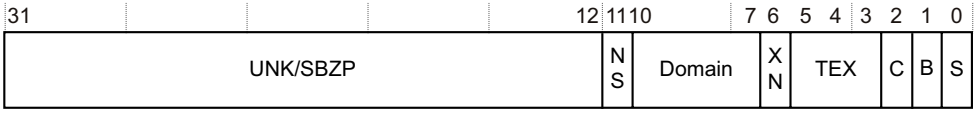


Figure 4-21 Main TLB Attributes Register bit assignments

Table 4-33 shows the TLB Attributes Register bit assignments. The Cortex-A9 processor does not support subpages.

Table 4-33 TLB Attributes Register bit assignments

Bits	Name	Description
[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.

4.3 CP14 Jazelle registers

In the Cortex-A9 implementation of the Jazelle Extension:

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

Table 4-34 shows the CP14 Jazelle registers. All Jazelle registers are 32 bits wide.

Table 4-34 CP14 Jazelle registers summary

CRn	Op1	CRm	Op2	Name	Type	Reset	Page
0	7	0	0	JIDR	RW[a]	`0xF4100168`	page 4-44
7	1	0	0	JOSCR	RW	-	page 4-45
7	2	0	0	JMCR	RW	-	page 4-47
7	3	0	0	Jazelle Parameters Register	RW	-	page 4-49
7	4	0	0	Jazelle Configurable Opcode Translation Table Register	WO	-	page 4-50

a. See *Write operation of the JIDR* on page 4-45 for the effect of a write operation

See the *ARM Architecture Reference Manual* for details of the Jazelle Extension.

4.4 CP14 Jazelle register descriptions

The following sections describe the CP14 Jazelle DBX registers arranged in numerical order, as shown in Table 4-34 on page 4-43:

- *Jazelle Identity and Miscellaneous Functions Register*
- *Jazelle Operating System Control Register* on page 4-45
- *Jazelle Main Configuration Register* on page 4-47
- *Jazelle Parameters Register* on page 4-49
- *Jazelle Configurable Opcode Translation Table Register* on page 4-50.

4.4.1 Jazelle Identity and Miscellaneous Functions 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 4-45.

Configurations Available in all configurations.

Attributes See the register summary in Table 4-34 on page 4-43.

Figure 4-22 shows the JIDR bit assignments.

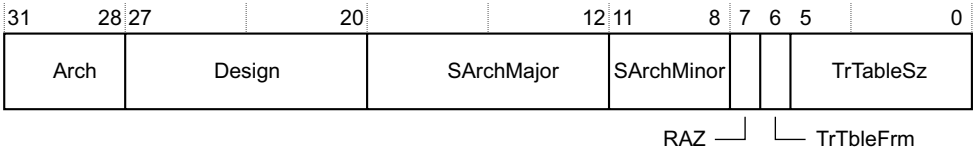


Figure 4-22 JIDR bit assignment

Table 4-35 shows the JIDR bit assignments.

Table 4-35 JIDR bit assignments

Bits	Name	Description
[31:28]	Arch	This uses the same architecture code that appears in the Main ID register.
[27:20]	Design	Contains the implementor 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, use:

```
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 4-50.

4.4.2 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 4-34 on page 4-43.

Figure 4-23 on page 4-46 shows the JOSCR bit assignments.

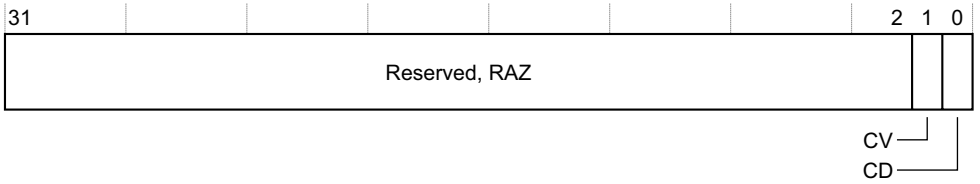


Figure 4-23 JOSCR bit assignments

Table 4-36 shows the JOSCR bit assignments.

Table 4-36 JOSCR bit assignments

Bits	Name	Description
[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, use:

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

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

4.4.3 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 4-34 on page 4-43.

Figure 4-24 shows the JMCR bit assignments.

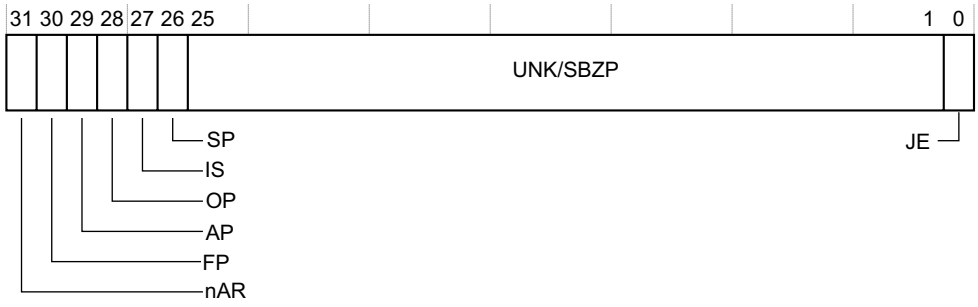


Figure 4-24 JMCR bit assignments

Table 4-37 shows the JMCR bit assignments.

Table 4-37 JMCR bit assignments

Bits	Name	Description
[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.
[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.

Table 4-37 JMCR bit assignments (continued)

Bits	Name	Description
[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, use:

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

4.4.4 Jazelle Parameters Register

The Jazelle Parameters Register characteristics are:

Purpose Describes the parameters that configure how the Jazelle hardware behaves.

Usage constraints Only accessible in privileged modes.

Configurations Available in all configurations.

Attributes See the register summary in Table 4-34 on page 4-43.

Figure 4-25 shows the Jazelle Parameters Register bit assignments.

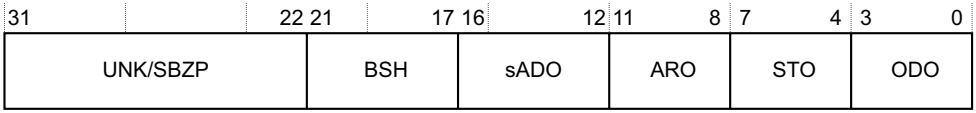


Figure 4-25 Jazelle Parameters Register bit assignments

Table 4-38 shows the Jazelle Parameters Register bit assignments.

Table 4-38 Jazelle Parameters Register bit assignments

Bits	Name	Description
[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, use:

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

4.4.5 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 4-34 on page 4-43.

Figure 4-26 shows the Jazelle Configurable Opcode Translation Table Register bit assignments.

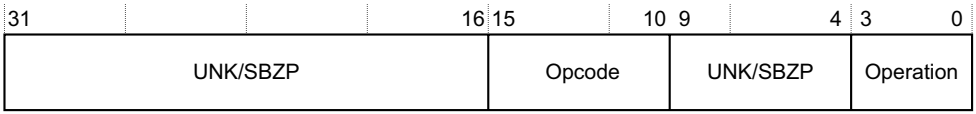


Figure 4-26 Jazelle Configurable Opcode Translation Table Register bit assignments

Table 4-39 shows the Jazelle Configurable Opcode Translation Table Register bit assignments.

Table 4-39 Jazelle Configurable Opcode Translation Table Register bit assignments

Bits	Name	Description
[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 this register, use:

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

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

Chapter 5
Memory Management Unit

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

5.1 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 application-specific 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 held in instruction and data TLBs.

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
  - support for 4 lockable entries using the lock-by-entry model
  - pseudo round-robin replacement policy
  - supports hardware page table walks to perform look-ups in the L1 data cache.

5.1.1 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.

Page sizes

The Cortex-A9 processor supports the following page sizes:

- 16MB supersections

  The processor supports supersections that consist of 16MB blocks of memory. The processor does not support the optional extension of physical address bits [39:32].
- 1MB sections
- 64KB large pages
- 4KB small pages.

Domains

Sixteen access domains are supported.

TLB

A two-level TLB structure is implemented. Four entries in the main TLB are lockable.

ASIDs

TLB entries can be global, or can be associated with particular processes or applications using ASIDs. ASIDs enable TLB entries to remain resident during context switches, avoiding the requirement of reloading them subsequently.

System control coprocessor

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

5.2 TLB Organization

TLB organization is described in the following sections:

- *Micro TLB*
- *Main TLB*.

5.2.1 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 look-up of the virtual addresses in a 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.

5.2.2 Main TLB

The main TLB is the second layer in the TLB structure that catches the misses from the Micro TLBs. It also provides a centralized source for lockable translation entries.

Misses from the instruction and data micro TLBs are handled by a unified main TLB. 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 two-way associative structure on 2x32or 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 currently 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. A TLB can store entries based on the following block sizes:

Supersections Describe 16MB blocks of memory.

Sections Describe 1MB blocks of memory.

Large pages Describe 64KB blocks of memory.

Small pages Describe 4KB blocks of memory.

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 within 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 Register* on page 4-32 for more information.

5.3 Memory Access Sequence

When the processor generates a memory access, the MMU:

1. Performs a look-up 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 look-up for the requested virtual address and current ASID and security state in the main TLB.
3. If there is a miss in 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 look-up 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 currently 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.

5.4 MMU interaction with the memory system

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

5.5 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.

5.5.1 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.

5.5.2 Synchronous and asynchronous aborts

Chapter 4 *The System Control Coprocessors* describes synchronous and asynchronous aborts, their priorities, and the IFSR and DFSR. 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 6
Level 1 Memory System

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

6.1 About the L1 memory system

The 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, and BTAC before using them. You are not required to invalidate the main TLB, even though it is recommended for safety reasons. This ensures compatibility with future revisions of the processor.

6.1.1 Memory system

This section describes:

- *Cache features*
- *Store buffer* on page 6-3.

Cache features

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

- Each cache can be disabled independently, using the system control coprocessor. See *System Control Register* on page 4-13.
- Cache replacement policy is either pseudo round-robin or pseudo random.
- 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.

- For optimum area and performance, all of the cache RAMs, and the associated tag RAMs, are designed to be implemented using standard ASIC RAM compilers.
- 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 is virtually indexed and physically tagged.

Data cache features

The data cache is physically indexed and physically tagged.

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 CPU has a store buffer with four 64-bit slots with data merging capability.

6.2 Security extensions support

The Cortex-A9 processor supports the TrustZone architecture, and exports the Secure or Non-secure status of its memory requests to the memory system.

6.3 About the L1 instruction side memory system

The L1 instruction side memory system is responsible for providing 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 6-1 shows this.

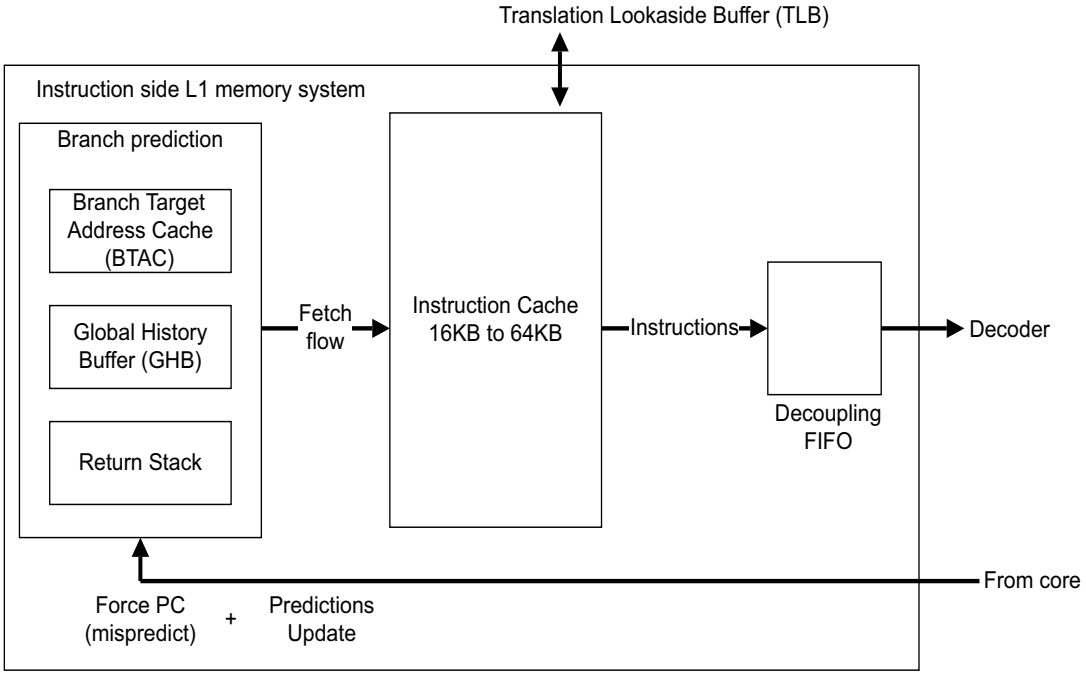


Figure 6-1 Branch prediction and instruction cache

The ISide comprises the following:

The Prefetch Unit (PFU)

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

- a two-way BTAC of 512 entries organized as two-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. Nor does it predict any instruction that changes the mode of the core. See *Program flow prediction*.

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 so as to provide up to four instructions per cycle to the prefetch unit
- security extensions support
- no lockdown support.

6.3.1 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 4-13. 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.

6.3.2 Program flow prediction

The following sections describe program flow prediction:

- *Predicted and non-predicted instructions* on page 6-7
- *Thumb state conditional branches* on page 6-7
- *Return stack predictions* on page 6-7.

Predicted and non-predicted 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 as 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 ARM or Thumb 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.

6.4 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.

6.4.1 Internal exclusive monitor

The Cortex-A9 processor L1 memory system has an internal exclusive monitor. This is a two-state, open and exclusive, state machine that manages load/store exclusive (`LDREXB`, `LDREXH`, `LDREX`, `LDREXD`, `STREXB`, `STREXH`, `STREX` and `STREXD`) accesses and clear exclusive (`CLREX`) instructions. You can use these instructions to construct semaphores, ensuring synchronization between different processes running on the CPU, 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 10-14

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 eight-word boundary. This size is implementation defined, and as such, software must not rely on this granularity remaining constant on other ARM cores.

6.4.2 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.

6.5 Data prefetching

This section describes:

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

6.5.1 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

6.5.2 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 4-17.

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.

6.6 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 6-2 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.

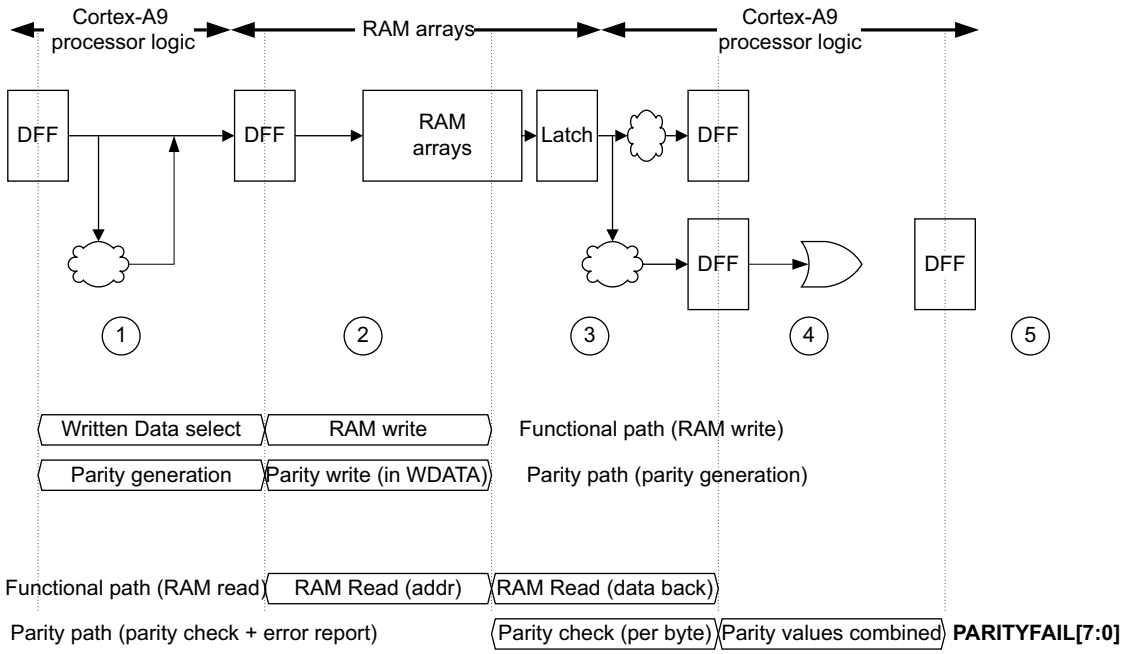


Figure 6-2 Parity support

The output signals **PARITYFAIL[7:0]** report parity errors. Typically, **PARITYFAIL[7:0]** reports parity errors 3 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.

6.6.1 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 7
Level 2 Memory Interface

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

7.1 Cortex-A9 L2 interface

This section describes the Cortex-A9 Level 2 interface in:

- *About the Cortex-A9 L2 interface*
- *Supported AXI transfers* on page 7-3
- *AXI transaction IDs* on page 7-4
- *STRT instructions* on page 7-9.

7.1.1 About the 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 7-1 shows the AXI master 0 interface attributes.

Table 7-1 AXI master 0 interface attributes

Attribute	Format
Write issuing capability	12, including: • eight noncacheable writes • four evictions
Read issuing capability	7, including: • six linefill reads. or • one noncacheable read
Combined issuing capability	19
Write ID capability	2
Write interleave capability	1
Write ID width	2
Read ID capability	3
Read ID width	2

Table 7-2 shows the AXI master 1 interface attributes.

Table 7-2 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
Write ID width	None
Read ID capability	4
Read ID width	2

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*.

Supported AXI transfers

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

For write-back write-allocate transfers the supported transfers are:

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

For noncacheable transactions:

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

7.1.2 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.

7.1.3 AXI USER bits

The AXI USER bits encodings are as follows:

Data side read bus, ARUSERM0[6:0]

Table 7-3 shows the bit encodings for **ARUSERM0[6:0]**

Table 7-3 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	b0 Non-shared b1 Shared

Instruction side read bus, ARUSERM1[6:0]

Table 7-4 shows the bit encodings for **ARUSERM1[6:0]**.

Table 7-4 ARUSERM1[6:0] encodings

Bits	Name	Description
[6]	Reserved	b0
[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	b0 Non-shared b1 Shared

Data side write bus, AWUSERM0[8:0]

Table 7-5 shows the bit encodings for **AWUSERM0[8:0]**.

Table 7-5 ARUSERM0[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 7-7.
[7]	Full line of write zeros bit	Indicates that the access is an entire cache line write full of zeros. See *Write full line of zeros* on page 7-8.
[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	b0 Non-shared b1 Shared

7.1.4 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.

7.2 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 PL310 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 7-8.
- *Speculative coherent requests* on page 7-8.

7.2.1 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 which 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 8 *Preload Engine*.

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

Note

No additional programming of the PL310 is required.

7.2.2 Early BRESP

According to the AXI specification, **BRESP** answers on response channels must be returned to the master only once the last data has been sent by the master. Cortex-A9 processors can also deal with **BRESP** answers returned as soon as address has been accepted by the slave, regardless of whether data is sent or not. This enables the Cortex-A9 processor to provide a higher bandwidth for writes if the slave can support the Early BRESP feature. Cortex-A9 processors set the **AWUSER[8]** bit to indicate to

the slave that it can accept an early **BRESP** answer for this access. This feature can optimize the performance of the processor, but the Early BRESP feature generates non-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 PL310 cache controller supports this non-AXI compliant feature.

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

Note

You must program the PL310 cache controller to benefit from this optimization. See the *PL310 Cache Controller TRM*.

7.2.3 Write full line of zeros

When this features is enabled, the Cortex-A9 processor can write entire non-coherent cache lines full of zero to the PL310 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 full line of write zeros by having the associated **AWUSER[7]** bit set.

Setting bit[3] of the ACTLR enables this feature. See *Auxiliary Control Register* on page 4-17.

You must program the PL310 Cache Controller first, prior to enabling the feature in the Cortex-A9 processor, to support this feature. See the *Pl310 Cache Controller TRM*.

7.2.4 Speculative coherent requests

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

7.3 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 the STRT and after the STRT.

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

Table 7-6 shows Cortex-A9 modes and corresponding **AxPROT** values.

Table 7-6 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 8
Preload Engine

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

8.1 About the Preload Engine

If implemented, the PLE loads selected regions of memory into L2 using an MCRR preload channel operation. New dedicated events monitor the behavior of the memory region. Additional PL310 events can also monitor PLE behavior.

Preload blocks enter the PLE FIFO. FIFO entries are 100 bits long and include:

- programmed parameters:
  - base address
  - length of stride
  - number of blocks.
- a valid bit
- an NS state bit
- a *Translation Table Base* (TTB) address, 30 bits long
- an *Address Space Identifier* (ASID) value, 8 bits long.

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 *Translation Lookaside Buffer* (TLB) maintenance operations, the maintenance operation must be applied to the FIFO entries too. 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.

8.2 PLE control register descriptions

This section describes the PLE control registers. The PLE control registers are:

- *PLE ID Register*
- *PLE Activity Status Register* on page 8-5
- *PLE FIFO Status Register* on page 8-6
- *Preload Engine User Accessibility Register* on page 8-7
- *Preload Engine Parameters Control Register* on page 8-8.

For all c11 system control registers NSAC.PLE controls Non-secure accesses. *PLE operations* on page 8-10 shows the operations to use with these control registers.

8.2.1 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 4-23 on page 4-33.

Figure 8-1 shows the PLEIDR bit assignments.

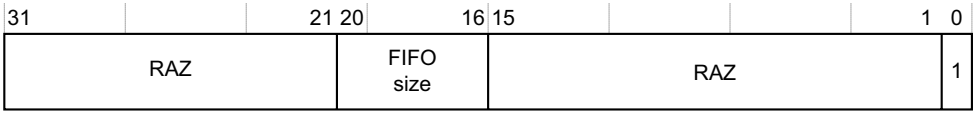


Figure 8-1 PLEIDR bit assignments

Table 8-1 shows the PLEIDR bit assignments.

Table 8-1 PLEIDR bit assignments

Bits	Name	Description
[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]	-	1 indicates that the Preload Engine is present in the given configuration.

To access the PLEIDR, use:

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

8.2.2 PLE Activity Status Register

The PLEASR characteristics are:

Purpose Indicates whether the PLE engine is currently 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 4-23 on page 4-33.

Figure 8-2 shows the PLEASR bit assignments.

Figure 8-2 PLEASR bit assignments

Table 8-2 shows the PLEASR bit assignments.

Table 8-2 PLEASR bit assignments

Bits	Name	Description
[31:1]	-	-
[0]	R	PLE Channel running 1 indicates that the Preload Engine is currently handling a PLE request.

To access the PLEASR, use:

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

8.2.3 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 4-23 on page 4-33.

Figure 8-3 shows the PLEFSR bit assignments.

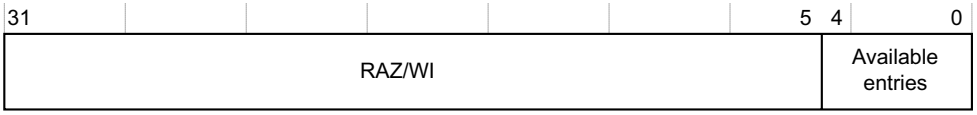


Figure 8-3 PLESFR bit assignments

Table 8-3 shows the PLEFSR bit assignments.

Table 8-3 PLESFR bit assignments

Bits	Name	Description
[31:5]	-	-
[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, which 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, use:

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

8.2.4 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 4-23 on page 4-33.

Figure 8-4 shows the PLEUAR bit assignments.

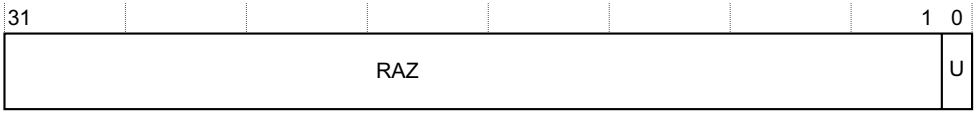


Figure 8-4 PLEUAR bit assignments

Table 8-4 shows the PLEUAR bit assignments.

Table 8-4 PLEUAR bit assignments

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

To access the PLEUAR, use:

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

8.2.5 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 4-23 on page 4-33.

Figure 8-5 shows the PLEPCR bit assignments.

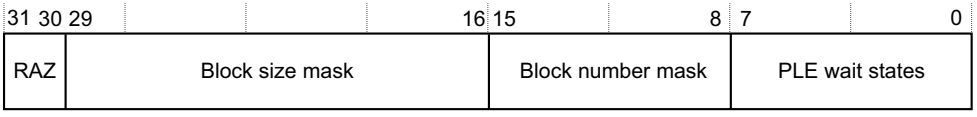


Figure 8-5 PLEPCR bit assignments

Table 8-5 shows the PLEPCR bit assignments.

Table 8-5 PLEPCR bit assignments

Bits	Name	Description
[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 = 8'b11111111, the PLE engine can issue one PLD request, a cache line, every 256 cycles. When PLE wait states = 8'b000000000, the PLE engine can issue one PLD request every cycle.

To access the PLEPCR, use:

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

8.3 PLE operations

The following sections describe the PLE operations:

- *Preload Engine FIFO flush operation*
- *Preload Engine pause channel operation*
- *Preload Engine resume channel operation* on page 8-11
- *Preload Engine kill channel operation* on page 8-11
- *PLE Program New Channel operation* on page 8-11.

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.

8.3.1 Preload Engine FIFO flush operation

The PLEFF operation characteristics are:

Purpose Flushes all PLE channels programmed previously including the PLE channel currently 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.

8.3.2 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 currently 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.

8.3.3 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
```

8.3.4 Preload Engine kill channel operation

The PLEKC operation characteristics are:

Purpose Kills the PLE channel currently active.

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
```

8.3.5 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 PLE channel currently active.

Figure 8-6 shows the <Rt>. and <Rt2> bit assignments for PLE program new channel operations.

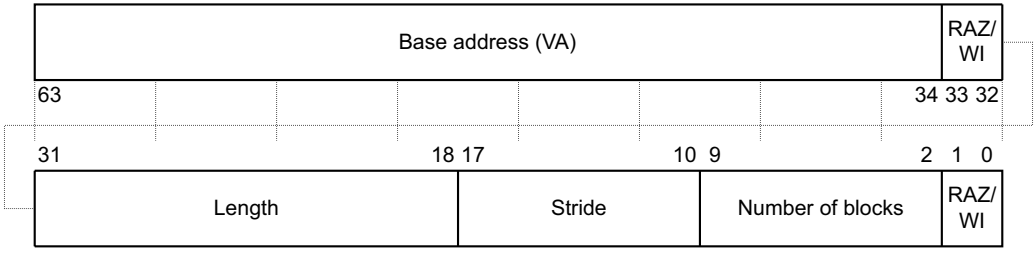


Figure 8-6 Program new channel operation bit assignments

Table 8-6 shows the PLE program new channel operation bit assignments.

Table 8-6 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 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 9-2 on page 9-4. See *PLE FIFO Status Register* on page 8-6.

Chapter 9
Performance Monitoring Unit

This chapter describes the *Performance Monitoring Unit* (PMU) and the registers that it can use. It contains the following sections:

9.1 About the Performance Monitoring Unit

The Cortex-A9 processor 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.

The PMU counters, and their associated control registers, are accessible from the internal CP15 interface as well as from the Debug APB interface.Table 9-1 shows the mappings of the PMU registers.

Table 9-1 Performance monitoring instructions and Debug APB mapping

Debug APB interface mapping	CP15 instruction	Access	Reset	Name
0x000	c9, 0, 13, 2	RW	-	PMXEVCNTR0
0x004	c9, 0, 13, 2	RW	-	PMXEVCNTR1
0x008	c9, 0, 13, 2	RW	-	PMXEVCNTR2
0x00C	c9, 0, 13, 2	RW	-	PMXEVCNTR3
0x010	c9, 0, 13, 2	RW	-	PMXEVCNTR4
0x014	c9, 0, 13, 2	RW	-	PMXEVCNTR5
0x07C	c9, 0, 13, 0	RW	-	PMCCNTR
0x400	c9, 0, 13, 1	RW	-	PMXEVTYPER0
0x404	c9, 0, 13, 1	RW	-	PMXEVTYPER1
0x408	c9, 0, 13, 1	RW	-	PMXEVTYPER2
0x40C	c9, 0, 13, 1	RW	-	PMXEVTYPER3
0x410	c9, 0, 13, 1	RW	-	PMXEVTYPER4
0x414	c9, 0, 13, 1	RW	-	PMXEVTYPER5
0xC00	c9, 0 12, 1	RW	0x00000000	PMCNTENSET
0xC20	c9, 0, 12, 2	RW	0x00000000	PMCNTENCLR
0xC40	c9, 0, 14, 1	RW	0x00000000	PMINTENSET
0xC60	c9, 0, 14, 2	RW	0x00000000	PMINTENCLR
0xC80	c9, 0, 12, 3	RW	-	PMOVSR
0xCA0	c9, 0, 12, 4	WO	-	PMSWINC

Table 9-1 Performance monitoring instructions and Debug APB mapping (continued)

Debug APB interface mapping	CP15 instruction	Access	Reset	Name
0xE04	c9, 0, 12, 0	RW	0x41093000	PMCR
0xE08	c9, 0, 14, 0	RW[a]	0x00000000	PMUSERENR
-	c9, 0, 12, 5	RW	-	PMSELR

a. Read only in user mode.

9.2 Performance monitoring events

The Cortex-A9 processor implements the architectural events described in the *ARM Architecture Reference Manual*, with the exception of:

0x08 Memory-reading 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-17.

The PMU provides an additional set of Cortex-A9 specific events.

9.2.1 Cortex-A9 specific events

Table 9-2 shows the Cortex-A9 specific events. In the value column of Table 9-2 Precise means the event is counted precisely. Events related to stalls and speculative instructions appear as Approximate entries in this column.

Table 9-2 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 which aborts late.	Approximate
0x50	Coherent linefill miss[b] Counts the number of coherent linefill requests performed by the Cortex-A9 processor which also miss in all the other Cortex-A9 processors, meaning 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 which hit in another Cortex-A9 processor, meaning 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, but does not receive any because of the instruction side not being able to provide any and the instruction cache is currently performing at least one linefill.	Approximate

Table 9-2 Cortex-A9 specific events (continued)

Event	Description	Value
0x61	Data cache dependent stall cycles Counts the number of cycles where the core 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 can be because of the instruction side not being able to provide the instructions, or to the data side not being able to provide the necessary data, because of them waiting for the main TLB translation table walk to complete.	Approximate
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 even more approximate 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. See Table A-17 on page A-17 for a description of how these values map to the PMUEVENT bus bits.	Approximate

Table 9-2 Cortex-A9 specific events (continued)

Event	Description	Value
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 which are flushed because of a previous load/store which 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
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. See Table A-17 on page A-17 for a description of how these values map to the PMUEVENT bus bits.	Approximate

Table 9-2 Cortex-A9 specific events (continued)

Event	Description	Value
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. See Table A-17 on page A-17 for a description of how these values map to the PMUEVENT bus bits.	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 and the data side is stalled too because it is full and executing 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 memory barrier. This includes all DMB instructions being executed, even speculatively.	Approximate
0x8A	Integer clock enabled Counts the number of cycles during which the integer core clock is enabled.	Approximate

Table 9-2 Cortex-A9 specific events (continued)

Event	Description	Value
0x8B	Data Engine clock enabled Counts the number of cycles during which the Data Engine clock is enabled.	Approximate
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 extensions. Otherwise reads as 0.
b. For use with Cortex-A9 multiprocessor variants.
c. Active only when the PLE is present. Otherwise reads as 0.

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 About the debug interface

The Cortex-A9 processor implements the ARMv7 debug architecture as described in the *ARM Architecture Reference Manual*. It implements the set of debug events 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 9-4.
- system coherency events.

See *Performance monitoring* on page 2-3. See also Chapter 9 *Performance Monitoring Unit*

10.1.1 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 10-23.

Note

The Cortex-A9 processor does not support Secure User Halting Debug Mode. That is, when **SPIDEN** is LOW, the core is not allowed to enter Halting Debug Mode even if the SUIDEN bit is set to 1. You can bypass this restriction by setting the external **SPIDEN** pin HIGH.

10.1.2 Breakpoints and watchpoints

There are:

- six breakpoints, two with Context ID comparison capability, BRP4 and BRP5. See *Breakpoint Value Registers* on page 10-8 and *Breakpoint Control Registers* on page 10-9.
- four watchpoints.

  A watchpoint event is always synchronous. It has the same behavior as a synchronous data abort. The method of debug entry (DSCR[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

See *Watchpoint Value Registers* on page 10-12 and *Watchpoint Control Registers* on page 10-13.

10.1.3 Asynchronous aborts

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

10.1.4 Processor interfaces

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

Debug registers

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

See *CTI signals* on page A-27 and *APB interface signals* on page A-26

Performance monitor

This interface is CP15 based and memory-mapped. See *Performance monitoring* on page 2-3. See also Chapter 9 *Performance Monitoring Unit*.

10.1.5 Effects of resets on debug registers

nDBGRESET

nDBGRESET 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.

10.2 About the Cortex-A9 debug interface

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 10-1 shows the Cortex-A9 debug registers interface.

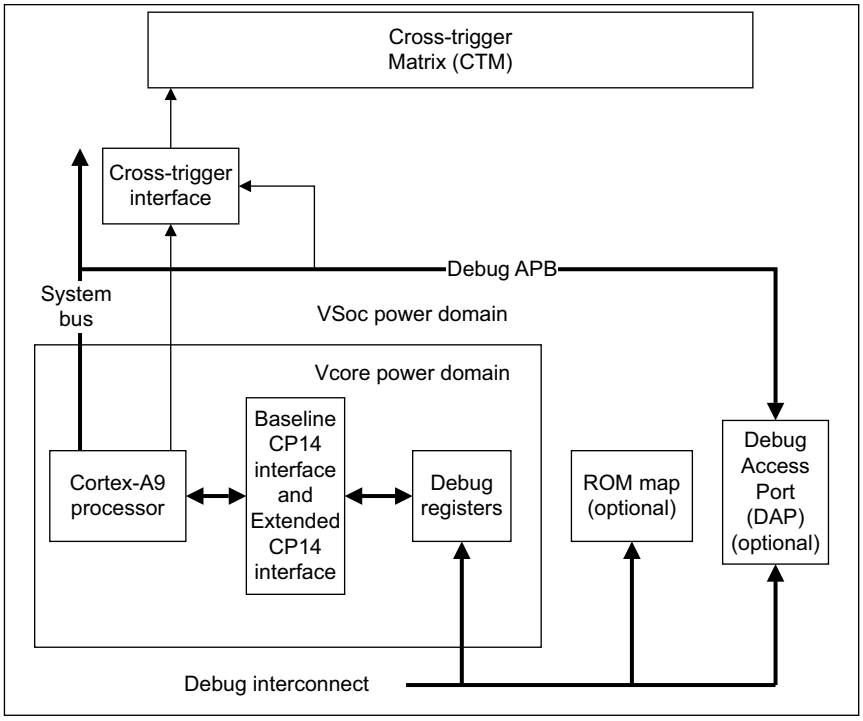


Figure 10-1 Debug registers interface

10.2.1 Debug register access

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, 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 interface registers

Register number	Offset	CP14 instruction	Access	Register name	Description
0	0x000	0 c0 c0 0	RO	DBGDIDR[a]	-[b]
-	-	0 c1 c0 0	RO	DBGDRAR[a]	-
-	-	0 c2 c0 0	RO	DBGDSAR[a]	-
-	-	0 c0 c1 0	RO	DBGDSCR-int[ab]	-
-	-	0 c0 c5 0	RW	DBGTR[a]	-
1-5	-	-	-	Reserved	-
6	0x018	0 c0 c6 0	RW	DBGWFAR	-
7	0x01C	0 c0 c7 0	RW	DBGVCR	-
8	-	-	-	Reserved	-
9	0x024	0 c0 c9 0	RAZ/WI	DBGECR	Not implemented
10	0x028	0 c0 c10 0	RAZ/WI	DBGDSCCR	*Debug State Cache Control Register (DBGDSCCR)* on page 10-8
11	0x02C	0 c0 c11 0	RAZ/WI	DBGDSMCR	Not implemented
12-31	-	-	-	Reserved	-
32	0x080	0 c0 c0 2	RW	DBGDTRRX -ext	-

Table 10-1 CP14 interface registers (continued)

Register number	Offset	CP14 instruction	Access	Register name	Description
33	0x084	0 c0 c1 2	WO	DBGITR	-
33	0x084	0 c0 c1 2	RO	DBGPCSR	-
34	0x088	0 c0 c2 2	RW	DBGDSCR-ext	-
35	0x08C	0 c0 c3 2	RW	DBGDTRTX-ext	-
36	0x090	0 c0 c4 2	WO	DBGDRCR	-
37-63	-	-	-	Reserved	-
64-79	0x100-0x13C	0 c0 c0 15 4	RW	DBGBVRn	*Breakpoint Value Registers* on page 10-8
80-95	0x140-0x17C	0 c0 c0 15 5	RW	DBGBCRn	*Breakpoint Control Registers* on page 10-9
96-111	0x180-0x1BC	0 c0 c0 15 6	RW	DBGWVRn	*Watchpoint Value Registers* on page 10-12
112-127	0x1C0-0x1FC	0 c0 c0 15 7	RW	DBGWCRn	*Watchpoint Control Registers* on page 10-13
128-191	-	-	-	Reserved	-
192	0x300	0 c1 c0 4	RAZ/WI	DBGOSLAR	Not implemented
193	0x304	0 c1 c1 4	RAZ/WI	DBGOSLSR	Not implemented
194	0x308	0 c1 c2 4	RAZ/WI	DBGOSSRR	Not implemented
195	-	-	-	Reserved	-
196	0x310	0 c1 c4 4	RO	DBGPRCR	-
197	0x314	0 c1 c5 4	RO	DBGPRSR	-
198-511	-	-	-	Reserved	-
512-575	0x800-0x8FC	-	-	-	PMU registers[c]
576-831	-	-	-	Reserved	-
832-895	0xD00-0xDFC	0 c6 c0 15 4-7	RW	Unpredictable	-
896-927	-	-		Reserved	-

Table 10-1 CP14 interface registers (continued)

Register number	Offset	CP14 instruction	Access	Register name	Description
928-959	0xE80-0xEFC0	0 c7 c0 15 2-3	RAZ/WI	-	-
960	0xF00	0 c7 c0 4	RAZ/WI	DBGITCTRL	Integration Mode Control Register
961-999	0xF04-0xF9C	-	-	-	-
1000	0xFA0	0 c7 c8 6	RW	DBGCLAIMSET	Claim Tag Set Register
1001	0xFA4	0 c7 c9 6	RW	DBGCLAIMCLR	Claim Tag Clear Register
1002-1003	-	-	-	Reserved	-
1004	0xFB0	0 c7 c12 6	WO	DBGLAR	Lock Access Register
1005	0xFB4	0 c7 c13 6	RO	DBGLSR	Lock Status Register
1006	0xFB8	0 c7 c14 6	RO	DBGAUTHSTATUS	Authentication Status Register
1007-1009	-	-	-	Reserved	-
1010	0xFC8	0 c7 c2 7	RAZ	DBGDEVID	-
1011	0xFCC	0 c7 c3 7	RO	DBGDEVTYPE	Device Type Register
1012-1016	0xFD0-0xFEC	0 c7 c4-8 7	RO	PERIPHERALID	*Identification Registers* on page 10-18
1017-1019	-	-	-	Reserved	-
1020-1023	0xFF0-0xFFC	0 c7 c12-15 7	RO	COMPONENTID	*Identification Registers* on page 10-18

a. Baseline CP14 interface. This register also has an external view through the memory-mapped interface and the CP14 interface.
b. Accessible in User mode if bit[12] of the DBGSCR is clear. Also accessible in privileged modes.
c. PMU registers are part of the CP15 interface. Reads from the extended CP14 interface return zero. See *c9 summary table* on page 4-31. See also Chapter 9 *Performance Monitoring Unit*.

10.3 Debug register descriptions

This section describes the debug registers.

10.3.1 Debug State Cache Control Register (DBGDSCCR)

The DSCCR controls cache behavior while the processor is in debug state. The Cortex-A9 processor does not implement any of the features of the Debug State Cache Control Register. The Debug State Cache Control Register reads as zero.

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

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.

10.3.3 Breakpoint Control Registers

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

- breakpoints
- linked breakpoints.

Figure 10-2 shows the bit arrangement of the BCRs.

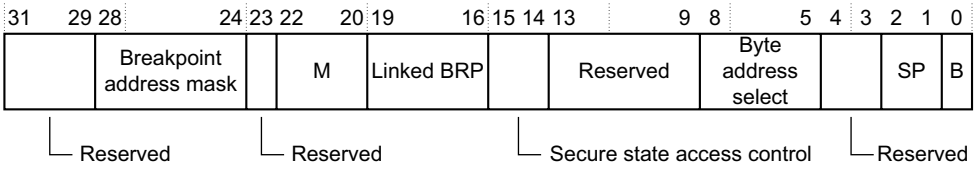


Figure 10-2 Breakpoint Control Registers bit assignments

Table 10-4 shows how the bit values correspond with the Breakpoint Control Registers functions.

Table 10-4 Breakpoint Control Registers 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.
[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.

Table 10-4 Breakpoint Control Registers bit assignments (continued)

Bits	Name	Description
[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: b0000 = the breakpoint never hits b0011 = the breakpoint hits if any of the two bytes starting at address BVR & `0xFFFFFFFC` +0 is accessed b1100 = the breakpoint hits if any of the two bytes starting at address BVR & `0xFFFFFFFC` +2 is accessed b1111 = the breakpoint 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 bits.

Table 10-5 Meaning of BVR 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.

10.3.4 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	**Register number**	**Offset**	**Register**	**Register number**	**Offset**
WVR0	96	0x180	WCR0	112	0x1C0
WVR1	97	0x184	WCR1	113	0x1C4
WVR2	98	0x188	WCR2	114	0x1C8
WVR3	99	0x18C	WCR3	115	0x1DC

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 BRP 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

10.3.5 Watchpoint Control Registers

The WCRs contain the necessary control bits for setting:

- watchpoints
- linked watchpoints.

Figure 10-3 on page 10-14 shows the bit arrangement of the Watchpoint Control Registers.

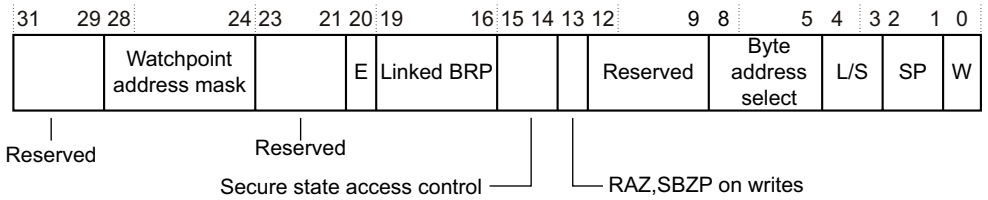


Figure 10-3 Watchpoint Control Registers bit assignments

Table 10-8 shows how the bit values correspond with the Watchpoint Control Registers functions.

Table 10-8 Watchpoint Control Registers 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.
[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.

Table 10-8 Watchpoint Control Registers bit assignments (continued)

Bits	Name	Description
[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.

10.4 Management registers

The Management registers define the standardized set of registers that is implemented by all CoreSight components. These registers are described in this section. The cp14 interface must be used to access these registers.

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

Table 10-9 Management registers

Offset	Register number	Access	Mnemonic	Description
0xD00-0xDFC	832-895	RO	-	*Processor ID Registers*.
0xE00-0xEF0	854-956	-	-	RAZ.
0xF00	960	RW	ITCTRL	-
0xF04-0xF9C	961-999	RAZ	-	Reserved for Management Register expansion.
0xFA0	1000	RW	CLAIMSET	-
0xFA4	1001	RW	CLAIMCLR	-
0xFA8-0xFBC	1002-1003	-	-	RAZ.
0xFB0	1004	WO	LOCKACCESS	-
0xFB4		RO	LOCKSTATUS	-
0xFB8		RO	AUTHSTATUS	-
0xFBC-0xFC4	1007-1009	-	-	RAZ.
0xFC8	1010	RO	DEVID	Device Identifier.
0xFCC	1011	RO	DEVTYPE	-
0xFD0-0xFFC	1012-1023	R	-	*Identification Registers* on page 10-18.

10.4.1 Processor ID Registers

The Processor ID Registers are read-only registers that return the same values as the corresponding CP15 ID Code Register and Feature ID Register.

Table 10-10 shows the offset value, register number, mnemonic, and description that are associated with each Process ID Register.

Table 10-10 Processor Identifier Registers

Offset (hex)	Register number	Mnemonic		Register value	Description
0xD00	832	CPUID	RO	Input dependent	ID Code Register
0xD04	833	CTYPR	RO	0x80038003	Cache Type Register
0xD08	834	-	RAZ	-	-
0xD0C	835	TTYPR	RO	0x00000400	TLB Type Register
0xD10-0xD1C	836-839	-	-	-	Reserved
0xD20	840	ID_PFR0	RO	0x00001231	Processor Feature Register 0
0xD24	841	ID_PFR1	RO	0x00000011	Processor Feature Register 1
0xD28	842	ID_DFR0	RO	0x00010444	Debug Feature Register 0
0xD2C	843	ID_AFR0	RAZ	-	Auxiliary Feature Register 0
0xD30	844	ID_MMFR0	RO	0x00100103	Memory Model Feature Register 0
0xD34	845	ID_MMFR1	RO	0x20000000	Memory Model Feature Register 1
0xD38	846	ID_MMFR2	RO	0x01230000	Memory Model Feature Register 2
0xD3C	847	ID_MMFR3	RO	0x00002111	Memory Model Feature Register 3
0xD40	848	ID_ISAR0	RO	0x00101111	Instruction Set Attribute Register 0
0xD44	849	ID_ISAR1	RO	0x13112111	Instruction Set Attribute Register 1
0xD48	850	ID_ISAR2	RO	0x21232041	Instruction Set Attribute Register 2
0xD4C	851	ID_ISAR3	RO	0x11112131	Instruction Set Attribute Register 3
0xD50	852	ID_ISAR4	RO	0x00011142	Instruction Set Attribute Register 4
0xD54	853	ID_ISAR5	RAZ	-	Instruction Set Attribute Register 5

10.4.2 Identification Registers

The Identification Registers are read-only registers that consist of the Peripheral Identification Registers and the Component Identification Registers. The Peripheral Identification Registers provide standard information required by all CoreSight components. Only bits [7:0] of each register are used.

The Component Identification Registers identify the processor as a CoreSight component. Only bits [7:0] of each register are used, the remaining bits Read-As-Zero. The values in these registers are fixed.

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

Table 10-11 Peripheral Identification Registers

Offset (hex)	Register number	Description
0xFD0	1012	Peripheral Identification Register 4
0xFD4	1013	Reserved
0xFD8	1014	Reserved
0xFDC	1015	Reserved
0xFE0	1016	Peripheral Identification Register 0
0xFE4	1017	Peripheral Identification Register 1
0xFE8	1018	Peripheral Identification Register 2
0xFEC	1019	Peripheral Identification Register 3

Table 10-12 shows fields that are in the Peripheral Identification Registers.

Table 10-12 Fields in the Peripheral Identification Registers

Field	Size	Description
4KB Count	4 bits	Indicates the Log_2 of the number of 4KB blocks occupied by the processor.
JEP106	4+7 bits	Identifies the designer of the processor. This field consists of a 4-bit continuation code and a 7-bit identity code.
Part number	12 bits	Indicates the part number of the processor.

Table 10-12 Fields in the Peripheral Identification Registers (continued)

Field	Size	Description
Revision	4 bits	Indicates the major and minor revision of the product. The major revision contains functionality changes and the minor revision contains bug fixes for the product.
RevAnd	4 bits	Indicates the manufacturer revision number. This number starts at `0x0` and increments by the integrated circuit manufacturer on metal fixes.
Customer modified	4 bits	-

Table 10-13 shows how the bit values correspond with the Peripheral ID Register 0 functions.

Table 10-13 Peripheral ID Register 0 bit functions

Bits	Description
[31:8]	RAZ.
[7:0]	Indicates bits [7:0] of the part number for the Cortex-A9 processor.

Table 10-14 shows how the bit values correspond with the Peripheral ID Register 1 functions.

Table 10-14 Peripheral ID Register 1 bit functions

Bits	Description
[31:8]	RAZ.
[7:4]	Indicates bits of the JEDEC JEP106 Identity Code.
[3:0]	Indicates bits [11:8] of the part number for the Cortex-A9 processor.

Table 10-15 shows how the bit values correspond with the Peripheral ID Register 2 functions.

Table 10-15 Peripheral ID Register 2 bit functions

Bits	Description
[31:8]	RAZ.
[7:4]	Indicates the revision number for the Cortex-A9 processor. This value changes based on the product major and minor revision.
[3]	-
[2:0]	Indicates bits [6:4] of the JEDEC JEP106 Identity Code.

Table 10-16 shows how the bit values correspond with the Peripheral ID Register 3 functions.

Table 10-16 Peripheral ID Register 3 bit functions

Bits	Description
[31:8]	RAZ.
[7:4]	Indicates the manufacturer revision number. This value changes based on the manufacturer metal fixes.
[3:0]	-

Table 10-17 shows how the bit values correspond with the Peripheral ID Register 4 functions.

Table 10-17 Peripheral ID Register 4 bit functions

Bits	Description
[31:8]	RAZ.
[7:4]	Indicates the number of blocks occupied by the Cortex-A9 processor.
[3:0]	Indicates the JEDEC JEP106 Continuation Code.

Table 10-18 shows the offset value, register number, and value that are associated with each Component Identification Register.

Table 10-18 Component Identification Registers

Offset (hex)	Register number	Value	Description
0xFF0	1020	0x0D	Component Identification Register 0
0xFF4	1021	0x90	Component Identification Register 1
0xFF8	1022	0x05	Component Identification Register 2
0xFFC	1023	0xB1	Component Identification Register 3

10.5 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 eleven 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 Performance Monitoring Unit (PMU) area of the Cortex-A9 processor. See Chapter 9 *Performance Monitoring Unit* 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 10-4 shows the external debug interface signals.

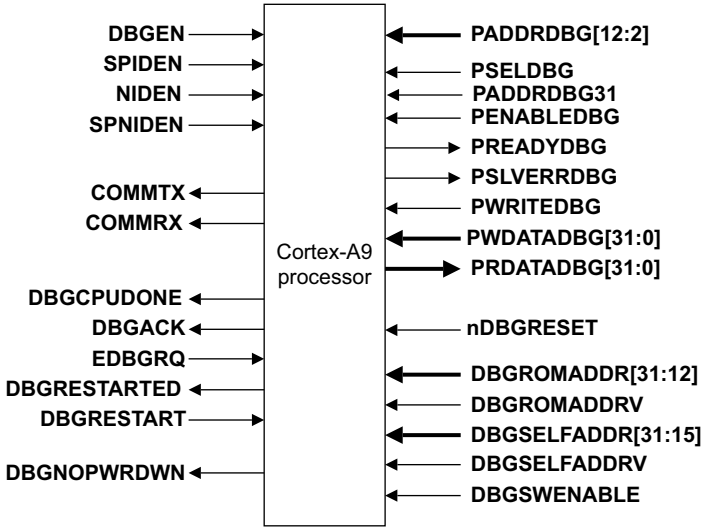


Figure 10-4 External debug interface signals

10.5.1 Authentication signals

Table 10-19 shows a list of the valid combinations of authentication signals along with their associated debug permissions.

Table 10-19 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 DSCR[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 core. For example, taking a breakpoint is defined as invasive debug but performance counters and trace are noninvasive.

10.5.2 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.

10.5.3 Debug APB interface

Table 10-20 shows the PMU register names and corresponding addresses on the Debug APB interface.

Table 10-20 PMU register names and Debug APB interface addresses

PMU register name	Debug APB Address
PMU event counter 0	`0x000`
PMU event counter 1	`0x004`
PMU event counter 2	`0x008`
PMU event counter 3	`0x00C`
PMU event counter 4	`0x010`

Table 10-20 PMU register names and Debug APB interface addresses (continued)

PMU register name	Debug APB Address
PMU event counter 5	0x014
pmccntr	0x07C
pmevtyper0	0x400
pmevtyper1	0x404
pmevtyper2	0x408
pmevtyper3	0x40C
pmevtyper4	0x410
pmevtyper5	0x414
pmcntenset	0xC00
pmcntenclr	0xC20
pmintenset	0xC40
pmintenclr	0xC60
pmovsr	0xC80
pmswinc	0xCA0
pmcr	0xE04
pmuserenr	0xE08

10.5.4 External debug request interface

The following sections describe the external debug request interface signals:

- *EDBGRQ* on page 10-26
- *DBGACK* on page 10-26
- *DBGCPUDONE* on page 10-26
- *COMMRX and COMMTX* on page 10-27
- *Memory mapped accesses, DBGROMADDR, and DBGSELFADDR* on page 10-28.

EDBGRQ

This signal generates a halting debug event, that is, it requests 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 core has completed a *Data Synchronization Barrier* (DSB).

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 10-5 on page 10-27 shows the Cortex-A9 connections specific to debug request and restart and the CoreSight pins.

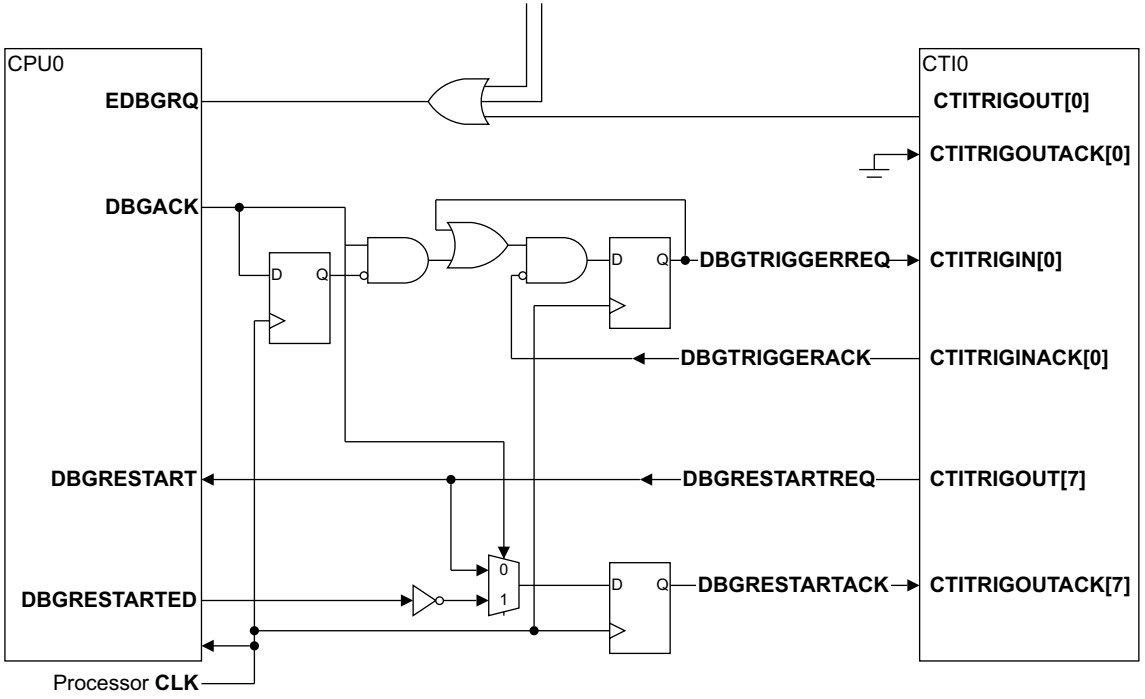


Figure 10-5 Debug request restart-specific connections

COMMRX and COMMTX

The **COMMRX** and **COMMTX** output signals enable interrupt-driven communications over the DTR. By connecting these signals to an interrupt controller, software using the debug communications channel can be interrupted whenever there is new data on the channel or when the channel is clear for transmission.

COMMRX is asserted when the CP14 DTR has data for the processor to read, and it is deasserted when the processor reads the data. Its value is equal to the DSCR[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 equals the inverse of the DSCR[29] DTRTX full flag.

Memory mapped accesses, DBGROMADDR, and DBGSELFADDR

Cortex-A9 processors have a memory-mapped debug interface. Cortex-A9 processors can access the debug and PMU registers by executing load and store instructions going through the AXI bus.

DBGROMADDR gives the base address for the ROM table which locates the physical addresses of the debug components.

DBGSELFADDR gives the offset from the ROM table to the physical addresses of the registers owned by the processor itself.

Appendix A
Signal Descriptions

This appendix lists and describes the Cortex-A9 signals. It contains the following sections:

A.1 Clock and clock control signals

The Cortex-A9 processor has a single externally generated global clock. Table A-1 shows the clock and clock control signal.

Table A-1 Clock and clock control signals for Cortex-A9

Name	I/O	Source	Description
CLK	I	Clock controller	Global clock. See *Clocking* on page 2-7.
MAXCLKLATENCY[2:0]	I	Implementation-specific static value	Controls dynamic clock gating delays. This pin is sampled during reset of the processor. See *Dynamic high level clock gating* on page 2-8

A.2 Resets and reset control

Table A-2 shows the reset and reset control signals.

Table A-2 Cortex-A9 processor 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 2-10.

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: 0 = EE bit is LOW 1 = EE bit is HIGH
CFGNMFI	I		Configures fast interrupts to be nonmaskable: 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. It sets the SCTLR.TE bit.
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`. It 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: 0 = not enabled 1 = enabled. See *System Control Register* on page 4-13.

A.5 Standby and Wait For Event signals

Table A-6 shows standby and wait for event signals.

Table A-6 Standby and wait for event signals

Name	I/O	Source or destination	Description
EVENTI	I	External coherent agent	Event input for Cortex-A9 processor wake-up from WFE state.
EVENTO	O		Event output. This signal is active HIGH during one CPU clock cycle when one `SEV` instruction is executed.
STANDBYWFI	O	Power controller	Indicates if the CPU is in WFI state: 0 = processor not in standby state 1 = processor in standby state.
STANDBYWFE	O		Indicates if the CPU is in WFE state: 0 = processor not in wait for event state 1 = processor in wait for event state.

See *Wait for interrupt (WFI/WFE) mode* on page 2-14.

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 2-12.

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*
- *AXI Master1 signals* on page A-13.

A.7.1 AXI Master0 signals

The following data read/write sections describe the AXI Master0 interface signals:

- *Write address signals for AXI Master0*
- *Write data channel signals* on page A-10
- *Write response channel signals* on page A-10
- *Read data channel signals* on page A-11
- *Read data channel signals* on page A-12
- *AXI Master0 Clock enable signals* on page A-13.

Write address signals for AXI Master0

Table A-8 shows the AXI write address signals for AXI Master0.

Table A-8 AXI-AW 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 b10 = WRAP Wrapping burst. All other values are reserved.
AWCACHEM0[3:0]	O		Cache type giving additional information about cacheable characteristics.
AWIDM0[1:0]	O		Request ID

Table A-8 AXI-AW 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. Each burst can be 1-16 transfers long: b0000 = 1 data transfer b0001 = 2 data transfers b0010 = 3 data transfers b0011 = 4 data transfers.
AWLOCKM0[1:0]	O		Lock type: b00 = normal access b01 = exclusive access b10 = locked access.
AWPROTM0[2:0]	O		Protection Type.
AWREADYM0	I		Address ready.
AWSIZEM0[1:0]	O		Burst 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 PL310. [7] full line of write zeros. Used with the PL310. [6] clean eviction. [5] level 1 eviction. [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.
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 signals for AXI Master0.

Table A-10 AXI-B 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 data channel signals

Table A-11 shows the AXI read address signals for AXI Master0.

Table A-11 AXI-AR 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. Each burst can be 1-16 transfers long: b0000 = 1 data transfer b0001 = 2 data transfers b0010 = 3 data transfers b0011 = 4 data transfers.
ARLOCKM0[1:0]	O		Lock type: b00 = normal access b01 = exclusive access b10 = locked access.
ARPROTM0[2:0]	O		Protection Type
ARREADYM0	I		Address ready.

Table A-11 AXI-AR signals for AXI Master0 (continued)

Name	I/O	Source or destination	Description
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] 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.
ARVALIDM0	O		Address valid.

Read data channel signals

Table A-12 shows the AXI read data signals for AXI Master0.

Table A-12 AXI-R 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 AXI Master0 clock enable signal

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* on page 2-7.

A.7.2 AXI Master1 signals

The following instruction interface sections describe the AXI Master1 interface signals:

- *Read data channel signals* on page A-14
- *Read data channel signals* on page A-15
- *AXI Master1 Clock enable signals* on page A-16.

Read data channel signals

Table A-14 shows the AXI read address signals for AXI Master1.

Table A-14 AXI-AR 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. Each burst can be 1-16 transfers long: b0000 = 1 data transfer b0001 = 2 data transfers b0010 = 3 data transfers b0011 = 4 data transfers.
ARLOCKM1[1:0]	O		Lock type: b00 = Normal access.
ARPROTM1[2:0]	O		Protection Type.
ARREADYM1	I		Address ready.

Table A-14 AXI-AR signals for AXI Master1 (continued)

Name	I/O	Destination	Description
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

This section describes the AXI Master1 clock enable signals. Table A-16 shows the AXI Master1 clock enable signals.

Table A-16 AXI Master1 clock enable signal

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* on page 2-7.

See Chapter 7 *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	Performance Monitoring Unit event bus. See Table A-18.
PMUIRQ	O		Performance Monitoring Unit 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]

Table A-18 Event signals and event numbers (continued)

Name	Event number	Description
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]
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

Table A-18 Event signals and event numbers (continued)

Name	Event number	Description
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
PMUEVENT[46]	0x8A	Integer core clock disabled
PMUEVENT[47]	0x8B	Data Engine clock disabled
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

Table A-18 Event signals and event numbers (continued)

Name	Event number	Description
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 9-4.

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 6-12.

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] D 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 r0p0 MBIST TRM* for a description of MBIST.

A.12 Scan test signal

Table A-24 lists 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-26
- *CTI signals* on page A-27
- *Miscellaneous debug interface signals* on page A-28.

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		Noninvasive debug enable: 0 = not enabled 1 = enabled.
SPIDEN	I		Secure privileged invasive debug enable: 0 = not enabled 1 = enabled.
SPNIDEN	I		Secure privileged noninvasive 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
PENABLEDBG	I	CoreSight APB devices	APB clock enable.
PRDATADBG[31:0]	O		APB read data bus.
PSELDBG	I		Debug registers select: 0 = debug registers not selected 1 = debug registers selected.
PSLVERRDBG	O		APB slave error signal.
PWRITEDBG	I		APB Read/Write signal.
PADDRDBG[12:2]	I		Programming address.
PADDRDBG31	I		APB address bus bit [31]: 0 = not an external debugger access 1 = external debugger access.
PREADYDBG	O		APB slave ready. An APB slave can assert **PREADY** to extend a transfer.
PWDATADBG[31:0]	I		APB write data.

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 performed by a debugger. Active HIGH.
DBGRESTART	I		Causes the core 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	Debugger has requested the Cortex-A9 processor is not powered down.
DBGSWENABLE	I	External debugger	When LOW only the external debug agent can modify debug registers. 0 = not enabled. 1 = enabled.
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 off to zero.
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 off to zero.
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 Input/Output column "I" indicates an input from the PTM interface to the Cortex-A9 processor. "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 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 core 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 and 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 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

Table A-29 PTM interface signals (continued)

Name	I/O	Source or destination	Description
WPTTRACEPROHIBITED	O	PTM device	Trace is prohibited for the current 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 core. 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 seen 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 current 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 the system control coprocessor* 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 2-5.

Appendix B
Instruction Cycle Timings

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-10.	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 andthe 64-bit alignment of the address. The resulting latency is the latency of the first loaded register. Table B-3 shows the cycle timings for load multiple operations.

Table B-3 Load multiple operations cycle timings

Instruction	AGU cycles to process the instruction		Resulting latency	
	Address aligned on a 64-bit boundary		Fast forward case	Other cases
	Yes	No		
LDM ,{1 register}	1	1	2	3
LDM ,{2 registers} 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-7 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.

Also, see *About the L1 instruction side memory system* on page 6-5 for some 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.To give useful cycle timing for these instructions is difficult because execution times are determined by the initial state of the processor.

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, which modifies the SPSR, is serializing:

- MSR SPSR.

Appendix C
Revisions

This appendix describes the technical changes between released issues of this book.s

Table C-1 Issue A

Change	Location
First release	-

Table C-2 Differences between issue A and issue B

Change	Location
Load/Store Unit and address generation clarified	Figure 1-1 on page 1-2.
Fast loop mode changed to small loop mode	• Figure 1-1 on page 1-2 • Small loop mode on page 1-3 • *Instruction cache features* on page 6-3 • About power consumption control on page 12-6.
"Branch prediction" changed to "dynamic branch prediction".	• *Features* on page 1-6 • *About the L1 instruction side memory system* on page 6-5 • *Branch instructions* on page B-9.
"LI cache coherency" changed to "L1 data cache coherency"	*Cortex-A9 variants* on page 1-4.
Processor Feature Register 0 reset value corrected	Table 4-29 on page 4-46.
PMSWINC descriptions made consistent	• Table 4-29 on page 4-46 • *Software Increment Register* on page 4-100.
MIDR bits[3:0] updated from 0 to 1	Table 4-1 on page 4-5.
ID_MMFR3 [23:20] bit value corrected to 0x1	Table 4-42 on page 4-50.
AFE bit description corrected	Table 4-51 on page 4-62.
Auxiliary Control Register bit field corrections	• Table 4-52 on page 4-66 • Figure 4-36 on page 4-66.
S parameter values corrected	Set/Way format on page 4-83.
Bit descriptions of bits[11], [10], and [8] made consistent with table	Figure 4-41 on page 4-87.
Description of event `0x68` corrected, "architecturally" removed.	Table 4-80 on page 4-123.
TLB lockdown entries number corrected from 8 to 4	c10, TLB Lockdown Register on page 4-134.
A,I, and F bit descriptions corrected	c12, Interrupt Status Register on page 4-147.
Number of micro TLB entries changed from 8 to 32	*Micro TLB* on page 5-4.
Repeated information about cache types removed	*Micro TLB* on page 5-4.

Table C-2 Differences between issue A and issue B (continued)

Change	Location
IRGN bits description amended from TTBCR to TTBR0/TTRBR1	*Main TLB* on page 5-4.
Note about invalidating the caches and BTAC before use added	*About the L1 memory system* on page 6-2.
Parity support scheme information section added	*Parity error support* on page 6-12.
L2 master interfaces, M0 and M1 listed and described	*About the Cortex-A9 L2 interface* on page 7-2.
Cross reference to DBSCR external description added. Footnote extended to include reference to the DBSCR external view	Table 10-1 on page 10-5.
DBGDSCR description corrected with the addition of internal and external view descriptions.	*CP14 c1, Debug Status and Control Register (DBGDSCR)* on page 8-9.
MOE bits descriptions re-ordered and extended	Table 8-2 on page 8-10.
Additional cross-references added from Table 10-1	• *Debug State Cache Control Register (DBGDSCCR)* on page 8-8 • *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.
Table 10-1 footnotes corrected	Table 10-1 on page 10-5.

Table C-2 Differences between issue A and issue B (continued)

Change	Location
Byte address field entries corrected.	Table 10-8 on page 10-14.
Interrupts signals descriptions corrected	Table A-3 on page A-4.
AXI USER descriptions extended	• Table A-8 on page A-8 • Table A-11 on page A-11 • Table A-14 on page A-14.

Table C-3 Differences between issue B and issue C

Change	Location
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** is removed.	-
Dynamic high level clock gating added.	*Dynamic high level clock gating* on page 2-9
TLB information updated.	Table 1-1 on page 1-10, Table 4-10 on page 4-15, Table 4-37 on page 4-44
ID_MMF3[15:12] description shortened.	*Memory Model Features Register 3* on page 4-49
ACTLR updated to include reference to PL310 optimizations.	*Auxiliary Control Register* on page 4-64
Addition of a second replacement strategy. Selection done by SCTLR.RR bit.	*System Control Register* on page 4-13
Event information extended.	*Cortex-A9 specific events* on page 4-32
DEFLAGS[6:0] added.	*DEFLAGS[6:0]* on page 4-37, *Performance monitoring signals* on page A-17
Power Control Register description added.	*Power Control Register* on page 4-63
PL310 optimizations added to L2 memory interface description	*Optimized accesses to the L2 memory interface* on page 7-7

Table C-3 Differences between issue B and issue C (continued)

Change	Location
Addition of watchpoint address masking	*Watchpoint Control Registers* on page 10-13
Added debug request restart diagram.	*Effects of resets on debug registers* on page 10-3
CPUCLKOFF information added.	Table A-4 on page A-5,*Unregistered signals* on page B-3
DECLKOFF information added.	Table A-4 on page A-5,*Unregistered signals* on page B-3
MAXCLKLATENCY[2:0] information added.	*Configuration signals* on page A-5
PMUEVENT bus description extended.	*Performance monitoring signals* on page A-17
PMUSECURE and **PMUPRIV** added.	*Performance monitoring signals* on page A-17
Description of serializing behavior of `DMB` updated.	*Serializing instructions* on page B-10

Table C-4 Differences between issue C and issue D

Change	Location
Preface updated	Preface
ARM Architecture Reference Manual moved to the top of Further Reading	*ARM publications* on page xx
Block diagram includes *Preload Engine* (PE)	Figure 1-1 on page 1-2
Interrupt signals amended	
Clarification of Data Engine options	*Data Engine* on page 1-2
Clarification of system design components	*System design components* on page 1-3
Compliance clarifications	*Compliance* on page 1-5
PE added to features	*Features* on page 1-6
Configurable options includes PE and PE FIFO size	*Configurable options for the Cortex-A9 processor* on page 1-8

Table C-4 Differences between issue C and issue D (continued)

Change	Location
NEON SIMD and FPU options clarified	Table 1-1 on page 1-8
Test Features section added	*Test features* on page 1-9
Power control description includes NEON SIMD clock gating	*Power Control Register* on page 2-8
nNEONRESET replaces **nDERESET**	*Reset modes* on page 2-10
nWDRESET added	
nPERIPHRESET added	
Changes to voltage domain boundaries	Figure 2-4 on page 2-17
Content of 4.1 that duplicates *ARM Architecture Reference Manual* material removed	
Sentence about tying unused bits of PARITYFAIL HIGH removed	*Parity error support* on page 6-12
PE description added	Chapter 8 *Preload Engine*
PMU description added	Chapter 9 *Performance Monitoring Unit*
Debug updated	on page 10-1Chapter 10 *Debug*
Signals descriptions amended and extended	Appendix A *Signal Descriptions*
AC Characteristics Appendix removed	

Table C-5 Differences between issue D and issue E

Change	Location
No technical changes	-

Glossary

This glossary describes some of the terms used in ARM manuals. Where terms can have several meanings, the meaning presented here is intended.

Abort
A mechanism that indicates to a core that the value associated with a memory access is invalid. An abort can be caused by the external or internal memory system as a result of attempting to access invalid instruction or data memory. An abort is classified as either a Prefetch or Data Abort, and an internal or External Abort.

See also Data Abort, External Abort and Prefetch Abort.

Abort model
An abort model is the defined behavior of an ARM processor in response to a Data Abort exception. Different abort models behave differently with regard to load and store instructions that specify base register write-back.

Addressing modes
A mechanism, shared by many different instructions, for generating values used by the instructions. For four of the ARM addressing modes, the values generated are memory addresses (the traditional role of an addressing mode). A fifth addressing mode generates values to be used as operands by data-processing instructions.

Advanced eXtensible Interface (AXI)
A bus protocol that supports separate address/control and data phases, unaligned data transfers using byte strobes, burst-based transactions with only start address issued, separate read and write data channels to enable low-cost DMA, ability to issue multiple

outstanding addresses, out-of-order transaction completion, and easy addition of register stages to provide timing closure.The AXI protocol also includes optional extensions to cover signaling for low-power operation.

AXI is targeted at high performance, high clock frequency system designs and includes a number of features that make it very suitable for high speed sub-micron interconnect.

Advanced High-performance Bus (AHB)
A bus protocol with a fixed pipeline between address/control and data phases. It only supports a subset of the functionality provided by the AMBA AXI protocol. The full AMBA AHB protocol specification includes a number of features that are not commonly required for master and slave IP developments and ARM Limited recommends only a subset of the protocol is usually used. This subset is defined as the AMBA AHB-Lite protocol.

See also Advanced Microcontroller Bus Architecture and AHB-Lite.

Advanced Microcontroller Bus Architecture (AMBA)
A family of protocol specifications that describe a strategy for the interconnect. AMBA is the ARM open standard for on-chip buses. It is an on-chip bus specification that describes a strategy for the interconnection and management of functional blocks that make up a *System-on-Chip* (SoC). It aids in the development of embedded processors with one or more CPUs or signal processors and multiple peripherals. AMBA complements a reusable design methodology by defining a common backbone for SoC modules.

Advanced Peripheral Bus (APB)
A simpler bus protocol than AXI and AHB. It is designed for use with ancillary or general-purpose peripherals such as timers, interrupt controllers, UARTs, and I/O ports. Connection to the main system bus is through a system-to-peripheral bus bridge that helps to reduce system power consumption.

AHB *See* Advanced High-performance Bus.

AHB Access Port (AHB-AP)
An optional component of the DAP that provides an AHB interface to a SoC.

AHB-AP *See* AHB Access Port.

AHB-Lite A subset of the full AMBA AHB protocol specification. It provides all of the basic functions required by the majority of AMBA AHB slave and master designs, particularly when used with a multi-layer AMBA interconnect. In most cases, the extra facilities provided by a full AMBA AHB interface are implemented more efficiently by using an AMBA AXI protocol interface.

Aligned A data item stored at an address that is divisible by the number of bytes that defines the data size is said to be aligned. Aligned words and halfwords have addresses that are divisible by four and two respectively. The terms word-aligned and halfword-aligned therefore stipulate addresses that are divisible by four and two respectively.

AMBA *See* Advanced Microcontroller Bus Architecture.

Advanced Trace Bus (ATB)
A bus used by trace devices to share CoreSight capture resources.

APB *See* Advanced Peripheral Bus.

Architecture The organization of hardware and/or software that characterizes a processor and its attached components, and enables devices with similar characteristics to be grouped together when describing their behavior, for example, Harvard architecture, instruction set architecture, ARMv6 architecture.

ARM instruction A word that specifies an operation for an ARM processor to perform. ARM instructions must be word-aligned.

ARM state A processor that is executing ARM (32-bit) word-aligned instructions is operating in ARM state.

ATB *See* Advanced Trace Bus.

ATB bridge A synchronous ATB bridge provides a register slice to facilitate timing closure through the addition of a pipeline stage. It also provides a unidirectional link between two synchronous ATB domains.

An asynchronous ATB bridge provides a unidirectional link between two ATB domains with asynchronous clocks. It is intended to support connection of components with ATB ports residing in different clock domains.

ATPG *See* Automatic Test Pattern Generation.

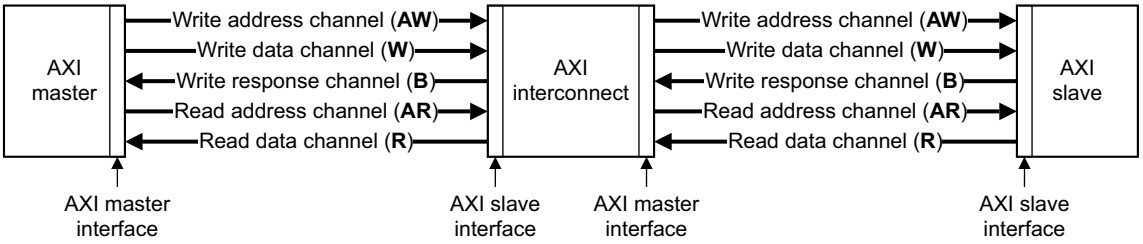
Automatic Test Pattern Generation (ATPG)
The process of automatically generating manufacturing test vectors for an ASIC design, using a specialized software tool.

AXI *See* Advanced eXtensible Interface.

AXI channel order and interfaces
The block diagram shows:

- the order that AXI channel signals are described in

- the master and slave interface conventions for AXI components.



AXI terminology The following AXI terms are general. They apply to both masters and slaves:

Active read transaction

A transaction where the read address has transferred, but the last read data has not yet transferred.

Active transfer

A transfer where the **xVALID**[1] handshake has asserted, but **xREADY** has not yet asserted.

Active write transaction

A transaction where the write address or leading write data has transferred, but the write response has not yet transferred.

Completed transfer

A transfer where the **xVALID**/**xREADY** handshake is complete.

Payload The non-handshake signals in a transfer.

Transaction An entire burst of transfers, comprising an address, one or more data transfers and a response transfer (writes only).

Transmit An initiator driving the payload and asserting the relevant **xVALID** signal.

Transfer A single exchange of information. That is, with one **xVALID**/**xREADY** handshake.

---

1. The letter **x** in the signal name denotes an AXI channel as follows:

AW	Write address channel.
W	Write data channel.
B	Write response channel.
AR	Read address channel.
R	Read data channel.

The following AXI terms are master interface attributes. To obtain optimum performance, they must be specified for all components with an AXI master interface:

Combined issuing capability

The maximum number of active transactions that a master interface can generate. This is specified instead of write or read issuing capability for master interfaces that use a combined storage for active write and read transactions.

Read ID capability

The maximum number of different **ARID** values that a master interface can generate for all active read transactions at any one time.

Read ID width

The number of bits in the **ARID** bus.

Read issuing capability

The maximum number of active read transactions that a master interface can generate.

Write ID capability

The maximum number of different **AWID** values that a master interface can generate for all active write transactions at any one time.

Write ID width

The number of bits in the **AWID** and **WID** buses.

Write interleave capability

The number of active write transactions that the master interface is capable of transmitting data for. This is counted from the earliest transaction.

Write issuing capability

The maximum number of active write transactions that a master interface can generate.

The following AXI terms are slave interface attributes. To obtain optimum performance, they must be specified for all components with an AXI slave interface

Combined acceptance capability

The maximum number of active transactions that a slave interface can accept. This is specified instead of write or read acceptance capability for slave interfaces that use a combined storage for active write and read transactions.

Read acceptance capability

The maximum number of active read transactions that a slave interface can accept.

Read data reordering depth

The number of active read transactions that a slave interface can transmit data for. This is counted from the earliest transaction.

Write acceptance capability

The maximum number of active write transactions that a slave interface can accept.

Write interleave depth

The number of active write transactions that the slave interface can receive data for. This is counted from the earliest transaction.

Banked registers

Those physical registers whose use is defined by the current processor mode. The banked registers are r8 to r14.

Base register

A register specified by a load or store instruction that is used to hold the base value for the instruction's address calculation. Depending on the instruction and its addressing mode, an offset can be added to or subtracted from the base register value to form the virtual address that is sent to memory.

Base register write-back

Updating the contents of the base register used in an instruction target address calculation so that the modified address is changed to the next higher or lower sequential address in memory. This means that it is not necessary to fetch the target address for successive instruction transfers and enables faster burst accesses to sequential memory.

Beat

Alternative word for an individual transfer within a burst. For example, an INCR4 burst comprises four beats.

See also Burst.

BE-8

Big-endian view of memory in a byte-invariant system.

See also BE-32, LE, Byte-invariant and Word-invariant.

BE-32

Big-endian view of memory in a word-invariant system.

See also BE-8, LE, Byte-invariant and Word-invariant.

Big-endian Byte ordering scheme where bytes of decreasing significance in a data word are stored at increasing addresses in memory.

See also Little-endian and Endianness.

Big-endian memory Memory where:

- a byte or halfword at a word-aligned address is the most significant byte or halfword within the word at that address
- a byte at a halfword-aligned address is the most significant byte within the halfword at that address.

See also Little-endian memory.

Block address An address that comprises a tag, an index, and a word field. The tag bits identify the way that contains the matching cache entry for a cache hit. The index bits identify the set being addressed. The word field contains the word address that can be used to identify specific words, halfwords, or bytes within the cache entry.

See also Cache terminology diagram on the last page of this glossary.

Boundary scan chain

A boundary scan chain is made up of serially-connected devices that implement boundary scan technology using a standard JTAG TAP interface. Each device contains at least one TAP controller containing shift registers that form the chain connected between **TDI** and **TDO**, through which test data is shifted. Processors can contain several shift registers to enable you to access selected parts of the device.

Branch prediction The process of predicting if conditional branches are to be taken or not in pipelined processors. Successfully predicting if branches are to be taken enables the processor to prefetch the instructions following a branch before the condition is fully resolved. Branch prediction can be done in software or by using custom hardware. Branch prediction techniques are categorized as static, where the prediction decision is decided before run time, and dynamic, where the prediction decision can change during program execution.

Breakpoint A breakpoint is a mechanism provided by debuggers to identify an instruction that program execution is to be halted at. Breakpoints are inserted by the programmer to enable inspection of register contents, memory locations, variable values at fixed points in the program execution to test that the program is operating correctly. Breakpoints are removed after the program is successfully tested.

See also Watchpoint.

Burst
A group of transfers to consecutive addresses. Because the addresses are consecutive, there is no requirement to supply an address for any of the transfers after the first one. This increases the speed that the group of transfers can occur at. Bursts over AHB buses are controlled using the **HBURST** signals to specify if transfers are single, four-beat, eight-beat, or 16-beat bursts, and to specify how the addresses are incremented.

See also Beat.

Byte
An 8-bit data item.

Byte-invariant
In a byte-invariant system, the address of each byte of memory remains unchanged when switching between little-endian and big-endian operation. When a data item larger than a byte is loaded from or stored to memory, the bytes making up that data item are arranged into the correct order depending on the endianness of the memory access. The ARM architecture supports byte-invariant systems in ARMv6 and later versions. When byte-invariant support is selected, unaligned halfword and word memory accesses are also supported. Multi-word accesses are expected to be word-aligned.

See also Word-invariant.

Byte lane strobe
An AHB signal, **HBSTRB**, that is used for unaligned or mixed-endian data accesses to determine the byte lanes that are active in a transfer. One bit of **HBSTRB** corresponds to eight bits of the data bus.

Cache
A block of on-chip or off-chip fast access memory locations, situated between the processor and main memory, used for storing and retrieving copies of often used instructions and/or data. This is done to greatly reduce the average speed of memory accesses and so to increase processor performance.

See also Cache terminology diagram on the last page of this glossary.

Cache contention
When the number of frequently-used memory cache lines that use a particular cache set exceeds the set-associativity of the cache. In this case, main memory activity increases and performance decreases.

Cache hit
A memory access that can be processed at high speed because the instruction or data that it addresses is already held in the cache.

Cache line
The basic unit of storage in a cache. It is always a power of two words in size (usually four or eight words), and is required to be aligned to a suitable memory boundary.

See also Cache terminology diagram on the last page of this glossary.

Cache line index
The number associated with each cache line in a cache way. Within each cache way, the cache lines are numbered from 0 to (set associativity) -1.

See also Cache terminology diagram on the last page of this glossary.

Cache lockdown To fix a line in cache memory so that it cannot be overwritten. Enables critical instructions and/or data to be loaded into the cache so that the cache lines containing them are not subsequently reallocated. This ensures that all subsequent accesses to the instructions/data concerned are cache hits, and therefore complete as quickly as possible.

Cache miss A memory access that cannot be processed at high speed because the instruction/data it addresses is not in the cache and a main memory access is required.

Cache set A cache set is a group of cache lines (or blocks). A set contains all the ways that can be addressed with the same index. The number of cache sets is always a power of two.

See also Cache terminology diagram on the last page of this glossary.

Cache way A group of cache lines (or blocks). It is 2 to the power of the number of index bits in size.

See also Cache terminology diagram on the last page of this glossary.

Clean A cache line that has not been modified while it is in the cache is said to be clean. To clean a cache is to write dirty cache entries into main memory. If a cache line is clean, it is not written on a cache miss because the next level of memory contains the same data as the cache.

See also Dirty.

Clock gating Gating a clock signal for a macrocell with a control signal and using the modified clock that results to control the operating state of the macrocell.

Clocks Per Instruction (CPI)
See Cycles Per Instruction (CPI).

Coherency *See* Memory coherency.

Cold reset Also known as power-on reset. Starting the processor by turning power on. Turning power off and then back on again clears main memory and many internal settings. Some program failures can lock up the processor and require a cold reset to enable the system to be used again. In other cases, only a warm reset is required.

See also Warm reset.

Communications channel
The hardware used for communicating between the software running on the processor, and an external host, using the debug interface. When this communication is for debug purposes, it is called the Debug Comms Channel. In an ARMv6 compliant core, the communications channel includes the Data Transfer Register, some bits of the Data Status and Control Register, and the external debug interface controller, such as the DBGTAP controller in the case of the JTAG interface.

Condition field A four-bit field in an instruction that specifies a condition under which the instruction can execute.

Conditional execution
If the condition code flags indicate that the corresponding condition is true when the instruction starts executing, it executes normally. Otherwise, the instruction does nothing.

Context The environment that each process operates in for a multitasking operating system. In ARM processors, this is limited to mean the Physical Address range that it can access in memory and the associated memory access permissions.

Control bits The bottom eight bits of a Program Status Register (PSR). The control bits change when an exception arises and can be altered by software only when the processor is in a privileged mode.

Coprocessor A processor that supplements the main processor. It carries out additional functions that the main processor cannot perform. Usually used for floating-point math calculations, signal processing, or memory management.

Core reset *See* Warm reset.

CPI *See* Cycles per instruction.

CPSR *See* Current Program Status Register

Current Program Status Register (CPSR)
The register that holds the current operating processor status.

Cycles Per instruction (CPI)
Cycles per instruction (or clocks per instruction) is a measure of the number of computer instructions that can be performed in one clock cycle. This figure of merit can be used to compare the performance of different CPUs that implement the same instruction set against each other. The lower the value, the better the performance.

Data Abort An indication from a memory system to a core that it must halt execution of an attempted illegal memory access. A Data Abort is attempting to access invalid data memory.

See also Abort, External Abort, and Prefetch Abort.

Data cache A block of on-chip fast access memory locations, situated between the processor and main memory, used for storing and retrieving copies of often used data. This is done to greatly reduce the average speed of memory accesses and so to increase processor performance.

DBGTAP *See* Debug Test Access Port.

Debug Access Port (DAP)
A TAP block that acts as an AMBA (AHB or AHB-Lite) master for access to a system bus. The DAP is the term used to encompass a set of modular blocks that support system wide debug. The DAP is a modular component, intended to be extendable to support optional access to multiple systems such as memory mapped AHB and CoreSight APB through a single debug interface.

Debugger A debugging system that includes a program, used to detect, locate, and correct software faults, together with custom hardware that supports software debugging.

Direct-mapped cache
A one-way set-associative cache. Each cache set consists of a single cache line, so cache lookup selects and checks a single cache line.

Dirty A cache line in a write-back cache that has been modified while it is in the cache is said to be dirty. A cache line is marked as dirty by setting the dirty bit. If a cache line is dirty, it must be written to memory on a cache miss because the next level of memory contains data that has not been updated. The process of writing dirty data to main memory is called cache cleaning.

See also Clean.

DNM *See* Do Not Modify.

Do Not Modify (DNM)
In Do Not Modify fields, the value must not be altered by software. DNM fields read as Unpredictable values, and must only be written with the same value read from the same field on the same processor. DNM fields are sometimes followed by RAZ or RAO in parentheses to show the way the bits must read for future compatibility, but programmers must not rely on this behavior.

Doubleword A 64-bit data item. The contents are taken as being an unsigned integer unless otherwise stated.

Doubleword-aligned
A data item having a memory address that is divisible by eight.

EmbeddedICE logic An on-chip logic block that provides TAP-based debug support for ARM processor cores. It is accessed through the TAP controller on the ARM core using the JTAG interface.

EmbeddedICE-RT The JTAG-based hardware provided by debuggable ARM processors to aid debugging in real-time.

Endianness Byte ordering. The scheme that determines the order that successive bytes of a data word are stored in, in memory. An aspect of the system's memory mapping.

See also Little-endian and Big-endian

Exception A fault or error event that is considered serious enough to require that program execution is interrupted. Examples include attempting to perform an invalid memory access, external interrupts, and undefined instructions. When an exception occurs, normal program flow is interrupted and execution is resumed at the corresponding exception vector. This contains the first instruction of the interrupt handler to deal with the exception.

Exception service routine
See Interrupt handler.

Exception vector *See* Interrupt vector.

Exponent The component of a floating-point number that normally signifies the integer power to which two is raised in determining the value of the represented number.

External Abort An indication from an external memory system to a core that it must halt execution of an attempted illegal memory access. An External Abort is caused by the external memory system as a result of attempting to access invalid memory.

See also Abort, Data Abort and Prefetch Abort.

Flat address mapping
A system of organizing memory where each Physical Address contained within the memory space is the same as its corresponding Virtual Address.

Front of queue pointer
Pointer to the next entry to be written to in the write buffer.

Fully-associative cache
A cache that has only one cache set that consists of the entire cache. The number of cache entries is the same as the number of cache ways.

See also Direct-mapped cache.

Halfword A 16-bit data item.

Halting debug-mode One of two mutually exclusive debug modes. In Halting debug-mode a *debug event*, such as a a breakpoint or watchpoint, causes the processor to enter a special Debug state. In Debug state the processor is controlled through the external debug interface. This interface also provides access to all processor state, coprocessor state, memory and input/output locations.

See also Monitor debug-mode.

High vectors Alternative locations for exception vectors. The high vector address range is near the top of the address space, rather than at the bottom.

Host A computer that provides data and other services to another computer. Especially, a computer providing debugging services to a target being debugged.

IEEE 754 standard *IEEE Standard for Binary Floating-Point Arithmetic, ANSI/IEEE Std. 754-1985*. The standard that defines data types, correct operation, exception types and handling, and error bounds for floating-point systems. Most processors are built in compliance with the standard in either hardware or a combination of hardware and software.

IEM *See* Intelligent Energy Manager.

IGN *See* Ignore.

Ignore (IGN) Must ignore memory writes.

Illegal instruction An instruction that is architecturally Undefined.

Implementation-defined
Means that the behavior is not architecturally defined, but must be defined and documented by individual implementations.

Implementation-specific
Means that the behavior is not architecturally defined, and does not have to be documented by individual implementations. Used when there are a number of implementation options available and the option chosen does not affect software compatibility.

Imprecise tracing A filtering configuration where instruction or data tracing can start or finish earlier or later than expected. Most cases cause tracing to start or finish later than expected.

For example, if **TraceEnable** is configured to use a counter so that tracing begins after the fourth write to a location in memory, the instruction that caused the fourth write is not traced, although subsequent instructions are. This is because the use of a counter in the **TraceEnable** configuration always results in imprecise tracing.

Index *See* Cache index.

Index register A register specified in some load or store instructions. The value of this register is used as an offset to be added to or subtracted from the base register value to form the virtual address, which is sent to memory. Some addressing modes optionally enable the index register value to be shifted prior to the addition or subtraction.

Instruction cache A block of on-chip fast access memory locations, situated between the processor and main memory, used for storing and retrieving copies of often used instructions. This is done to greatly reduce the average speed of memory accesses and so to increase processor performance.

Instruction cycle count
The number of cycles that an instruction occupies the Execute stage of the pipeline for.

Intelligent Energy Manager (IEM)
A technology that enables dynamic voltage scaling and clock frequency variation to be used to reduce power consumption in a device.

Internal scan chain A series of registers connected together to form a path through a device, used during production testing to import test patterns into internal nodes of the device and export the resulting values.

Interrupt handler A program that control of the processor is passed to when an interrupt occurs.

Interrupt vector One of a number of fixed addresses in low memory, or in high memory if high vectors are configured, that contains the first instruction of the corresponding interrupt handler.

Invalidate To mark a cache line as being not valid by clearing the valid bit. This must be done whenever the line does not contain a valid cache entry. For example, after a cache flush all lines are invalid.

Joint Test Action Group (JTAG)
The name of the organization that developed standard IEEE 1149.1. This standard defines a boundary-scan architecture used for in-circuit testing of integrated circuit devices. It is commonly known by the initials JTAG.

JTAG *See* Joint Test Action Group.

LE Little endian view of memory in both byte-invariant and word-invariant systems. See also Byte-invariant, Word-invariant.

Line *See* Cache line.

Little-endian Byte ordering scheme where bytes of increasing significance in a data word are stored at increasing addresses in memory.

See also Big-endian and Endianness.

Little-endian memory
Memory where:

- a byte or halfword at a word-aligned address is the least significant byte or halfword within the word at that address
- a byte at a halfword-aligned address is the least significant byte within the halfword at that address.

See also Big-endian memory.

Load/store architecture
A processor architecture where data-processing operations only operate on register contents, not directly on memory contents.

Load Store Unit (LSU)
The part of a processor that handles load and store transfers.

LSU *See* Load Store Unit.

Macrocell A complex logic block with a defined interface and behavior. A typical VLSI system comprises several macrocells (such as a processor, an ETM, and a memory block) plus application-specific logic.

Memory bank One of two or more parallel divisions of interleaved memory, usually one word wide, that enable reads and writes of multiple words at a time, rather than single words. All memory banks are addressed simultaneously and a bank enable or chip select signal determines the bank that is accessed for each transfer. Accesses to sequential word addresses cause accesses to sequential banks. This enables the delays associated with accessing a bank to occur during the access to its adjacent bank, speeding up memory transfers.

Memory coherency A memory is coherent if the value read by a data read or instruction fetch is the value that was most recently written to that location. Memory coherency is made difficult when there are multiple possible physical locations that are involved, such as a system that has main memory, a write buffer and a cache.

Memory Management Unit (MMU)
Hardware that controls caches and access permissions to blocks of memory, and translates virtual addresses to physical addresses.

Memory Protection Unit (MPU)
Hardware that controls access permissions to blocks of memory. Unlike an MMU, an MPU does not translate virtual addresses to physical addresses.

Microprocessor *See* Processor.

Miss *See* Cache miss.

MMU *See* Memory Management Unit.

Monitor debug-mode
One of two mutually exclusive debug modes. In Monitor debug-mode the processor enables a software abort handler provided by the debug monitor or operating system debug task. When a breakpoint or watchpoint is encountered, this enables vital system interrupts to continue to be serviced while normal program execution is suspended.

See also Halt mode.

MPU *See* Memory Protection Unit.

VA *See* Modified Virtual Address.

PA *See* Physical Address.

Penalty The number of cycles in which no useful Execute stage pipeline activity can occur because an instruction flow is different from that assumed or predicted.

Power-on reset *See* Cold reset.

Prefetching In pipelined processors, the process of fetching instructions from memory to fill up the pipeline before the preceding instructions have finished executing. Prefetching an instruction does not mean that the instruction must be executed.

Prefetch Abort An indication from a memory system to a core that it must halt execution of an attempted illegal memory access. A Prefetch Abort can be caused by the external or internal memory system as a result of attempting to access invalid instruction memory.

See also Data Abort, External Abort and Abort.

Processor A processor is the circuitry in a computer system required to process data using the computer instructions. It is an abbreviation of microprocessor. A clock source, power supplies, and main memory are also required to create a minimum complete working computer system.

Physical Address (PA)

The MMU performs a translation on *Modified Virtual Addresses* (VA) to produce the *Physical Address* (PA) that is given to AXI to perform an external access. The PA is also stored in the data cache to avoid the necessity for address translation when data is cast out of the cache.

Read Reads are defined as memory operations that have the semantics of a load. That is, the ARM instructions LDM, LDRD, LDC, LDR, LDRT, LDRSH, LDRH, LDRSB, LDRB, LDRBT, LDREX, RFE, STREX, SWP, and SWPB, and the Thumb instructions LDM, LDR, LDRSH, LDRH, LDRSB, LDRB, and POP. Java bytecodes that are accelerated by hardware can cause a number of reads to occur, according to the state of the Java stack and the implementation of the Java hardware acceleration.

RealView ICE A system for debugging embedded processor cores using a JTAG interface.

Region A partition of instruction or data memory space.

Remapping Changing the address of physical memory or devices after the application has started executing. This is typically done to enable RAM to replace ROM when the initialization has been completed.

Reserved A field in a control register or instruction format is reserved if the field is to be defined by the implementation, or produces Unpredictable results if the contents of the field are not zero. These fields are reserved for use in future extensions of the architecture or are implementation-specific. All reserved bits not used by the implementation must be written as 0 and read as 0.

Saved Program Status Register (SPSR)
The register that holds the CPSR of the task immediately before the exception occurred that caused the switch to the current mode.

SBO *See* Should Be One.

SBZ *See* Should Be Zero.

SBZP *See* Should Be Zero or Preserved.

Scan chain A scan chain is made up of serially-connected devices that implement boundary scan technology using a standard JTAG TAP interface. Each device contains at least one TAP controller containing shift registers that form the chain connected between **TDI** and **TDO**, through which test data is shifted. Processors can contain several shift registers to enable you to access selected parts of the device.

SCREG The currently selected scan chain number in an ARM TAP controller.

Set *See* Cache set.

Set-associative cache
In a set-associative cache, lines can only be placed in the cache in locations that correspond to the modulo division of the memory address by the number of sets. If there are *n* ways in a cache, the cache is termed *n*-way set-associative. The set-associativity can be any number greater than or equal to 1 and is not restricted to being a power of two.

Should Be One (SBO)
Should be written as 1 (or all 1s for bit fields) by software. Writing a 0 produces Unpredictable results.

Should Be Zero (SBZ)
Should be written as 0 (or all 0s for bit fields) by software. Writing a 1 produces Unpredictable results.

Should Be Zero or Preserved (SBZP)
Should be written as 0 (or all 0s for bit fields) by software, or preserved by writing the same value back that has been previously read from the same field on the same processor.

SPSR *See* Saved Program Status Register

Standard Delay Format (SDF)
The format of a file that contains timing information to the level of individual bits of buses and is used in SDF back-annotation. An SDF file can be generated in a number of ways, but most commonly from a delay calculator.

Synchronization primitive
The memory synchronization primitive instructions are those instructions that are used to ensure memory synchronization. That is, the LDREX, STREX, SWP, and SWPB instructions.

Tag The upper portion of a block address used to identify a cache line within a cache. The block address from the CPU is compared with each tag in a set in parallel to determine if the corresponding line is in the cache. If it is, it is said to be a cache hit and the line can be fetched from cache. If the block address does not correspond to any of the tags, it is said to be a cache miss and the line must be fetched from the next level of memory.

See also Cache terminology diagram on the last page of this glossary.

TAP *See* Test access port.

Test Access Port (TAP)
The collection of four mandatory and one optional terminals that form the input/output and control interface to a JTAG boundary-scan architecture. The mandatory terminals are **TDI**, **TDO**, **TMS**, and **TCK**. The optional terminal is **TRST**. This signal is required in ARM cores because it is used to reset the debug logic.

Thumb instruction A halfword that specifies an operation for an ARM processor in Thumb state to perform. Thumb instructions must be halfword-aligned.

Thumb state A processor that is executing Thumb (16-bit) halfword aligned instructions is operating in Thumb state.

TLB *See* Translation Look-aside Buffer.

Translation Lookaside Buffer (TLB)
A cache of recently used page table entries that avoid the overhead of translation table walking on every memory access. Part of the Memory Management Unit.

Translation table A table, held in memory, that contains data that defines the properties of memory areas of various fixed sizes.

Translation table walk
The process of doing a full translation table lookup. It is performed automatically by hardware.

Trap An exceptional condition in a VFP coprocessor that has the respective exception enable bit set in the FPSCR register. The user trap handler is executed.

Undefined Indicates an instruction that generates an Undefined instruction trap. See the *ARM Architecture Reference Manual* for more details on ARM exceptions.

UNP *See* Unpredictable.

Unpredictable For reads, the data returned when reading from this location is unpredictable. It can have any value. For writes, writing to this location causes unpredictable behavior, or an unpredictable change in device configuration. Unpredictable instructions must not halt or hang the processor, or any part of the system.

Unsupported values

Specific data values that are not processed by the VFP coprocessor hardware but bounced to the support code for completion. These data can include infinities, NaNs, subnormal values, and zeros. An implementation is free to select which of these values is supported in hardware fully or partially, or requires assistance from support code to complete the operation. Any exception resulting from processing unsupported data is trapped to user code if the corresponding exception enable bit for the exception is set.

VA *See* Virtual Address.

Victim A cache line, selected to be discarded to make room for a replacement cache line that is required as a result of a cache miss. The method used to select the victim for eviction is processor-specific. A victim is also known as a cast out.

Virtual Address (VA)

The MMU uses its translation tables to translate a Virtual Address into a Physical Address. The processor executes code at the Virtual Address, possibly located elsewhere in physical memory.

See also Modified Virtual Address, and Physical Address.

Warm reset Also known as a core reset. Initializes the majority of the processor excluding the debug controller and debug logic. This type of reset is useful if you are using the debugging features of a processor.

Watchpoint A watchpoint is a mechanism provided by debuggers to halt program execution when the data contained by a particular memory address is changed. Watchpoints are inserted by the programmer to enable inspection of register contents, memory locations, and variable values when memory is written to test that the program is operating correctly. Watchpoints are removed after the program is successfully tested. *See also* Breakpoint.

Way *See* Cache way.

WB *See* Write-back.

Word A 32-bit data item.

Word-invariant In a word-invariant system, the address of each byte of memory changes when switching between little-endian and big-endian operation, in such a way that the byte with address A in one endianness has address A EOR 3 in the other endianness. As a result, each aligned word of memory always consists of the same four bytes of memory in the same order, regardless of endianness. The change of endianness occurs because of the change to the byte addresses, not because the bytes are rearranged.The ARM

architecture supports word-invariant systems in ARMv3 and later versions. When word-invariant support is selected, the behavior of load or store instructions that are given unaligned addresses is instruction-specific, and is in general not the expected behavior for an unaligned access. It is recommended that word-invariant systems use the endianness that produces the required byte addresses at all times, apart possibly from very early in their reset handlers before they have set up the endianness, and that this early part of the reset handler use only aligned word memory accesses.

See also Byte-invariant.

Write Writes are defined as operations that have the semantics of a store. That is, the ARM instructions SRS, STM, STRD, STC, STRT, STRH, STRB, STRBT, STREX, SWP, and SWPB, and the Thumb instructions STM, STR, STRH, STRB, and PUSH. Java bytecodes that are accelerated by hardware can cause a number of writes to occur, according to the state of the Java stack and the implementation of the Java hardware acceleration.

Write-back (WB) In a write-back cache, data is only written to main memory when it is forced out of the cache on line replacement following a cache miss. Otherwise, writes by the processor only update the cache. (Also known as copyback).

Write buffer A block of high-speed memory, arranged as a FIFO buffer, between the data cache and main memory, whose purpose is to optimize stores to main memory.

Write completion The memory system indicates to the processor that a write has been completed at a point in the transaction where the memory system is able to guarantee that the effect of the write is visible to all processors in the system. This is not the case if the write is associated with a memory synchronization primitive, or is to a Device or Strongly-ordered region. In these cases the memory system might only indicate completion of the write when the access has affected the state of the target, unless it is impossible to distinguish between having the effect of the write visible and having the state of target updated.

This stricter requirement for some types of memory ensures that any side-effects of the memory access can be guaranteed by the processor to have taken place. You can use this to prevent the starting of a subsequent operation in the program order until the side-effects are visible.

Write-through (WT) In a write-through cache, data is written to main memory at the same time as the cache is updated.

WT *See* Write-through.

Cache terminology diagram

The diagram illustrates the following cache terminology:

- block address

- cache line
- cache set
- cache way
- index

- tag.