Cortex™-A9

Revision: r1p0

Technical Reference Manual

ARM®

ARM DDI 0388C

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

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.

Restricted 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 manual, 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 enable 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 Coprocessor*

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 ***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 ***AC Characteristics***

Read this for a description of the Cortex-A9 AC characteristics.

Appendix C ***Instruction Cycle Timings***

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

Appendix D ***Revisions***

Read this for a description of technical changes in this document.

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

Conventions

Conventions that this book can use are described in:

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

Typographical

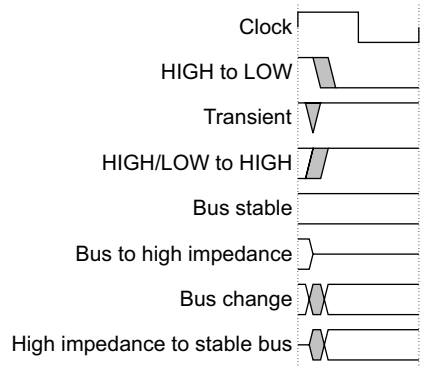
The typographical conventions are:

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 xxiii 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.

Further reading

This section lists publications by ARM and by third parties.

See `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:

- *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)
- *ARM Architecture Reference Manual, ARMv7-A and ARMv7-R edition* (ARM DDI 0406)
- *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 both on the Cortex-A9 processor, and on 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 manual

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.

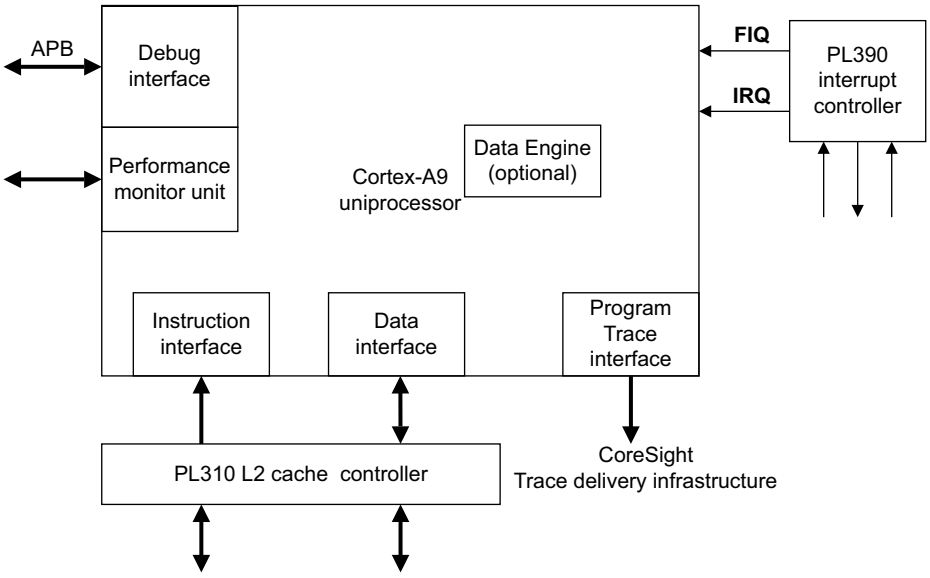


Figure 1-1 Cortex-A9 uniprocessor system.

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 *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.

If the design includes the MPE, the FPU is not included.

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

Floating-Point Unit

The *Floating-Point Unit* (FPU) is an ARMv7 VFPv3 D16 coprocessor without Advanced SIMD extensions (NEON). It is tightly integrated to the Cortex-A9 processor pipeline and uses both the Decode stage, and the load and store pipeline. 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.

The design can include the FPU only, in which case the MPE is not included.

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)* and the Cortex A9MP Interrupt Controller share the same programmers model. There are implementation-specific differences.

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

PrimeCell Level 2 Cache Controller (PL310)

The PrimeCell Level 2 Cache Controller reduces the number of external memory accesses.

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-A9MPCore 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-A9MPCore Technical Reference Manual* for more information.

The following system registers have Cortex-A9MPCore uses:

- *Multiprocessor Affinity Register* on page 4-37
- *Auxiliary Control Register* on page 4-63
- *Configuration Base Address Register* on page 4-126.

Some PMU event signals have Cortex-A9MPCore uses. See *Performance monitoring signals* on page A-16.

1.3 Compliance

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

- ARM Thumb®-2 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) architecture to enable execution environment acceleration. See the *ARM Architecture Reference Manual* for details of the ThumbEE instruction set.
- TrustZone® technology for enhanced security. See *Security extensions overview* on page 3-11. See the *ARM Reference Manual* for details on how TrustZone works in the architecture.
- NEON® technology to accelerate the performance of multimedia applications such as 3-D graphics and image processing. 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 Advanced SIMD (NEON) and 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 both Baseline CP14 and Extended CP14 debug access. To get full access to the processor debug capability, you can access the debug register map through the APB slave port. See Chapter 8 *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
  - asynchronous AXI wrappers
  - three voltage domains.
- optional VFPv3-D16 FPU with trapless execution
- optional Media Processing Engine with NEON technology
- optional Jazelle hardware acceleration.

1.5 Interfaces

The processor has the following external interfaces:

- AMBA AXI interfaces
- APB CoreSight 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	Selected by TLB_SIZE[a]
Jazelle Architecture Extension	Full or trivial	Full
Media Processing Engine with NEON technology	Included or not[b]	Not included
FPU	Included or not[c]	
PTM interface	Included or not	
IEM 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.

a. See *TLB Type Register* on page 4-36.
b. Includes support for floating-point operations. If this option is implemented then the FPU option cannot also be implemented.
c. If this option is implemented then the Media Processing Engine with NEON technology option cannot also be implemented.

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 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 *Further reading* on page xxv for more information about the books described in this section.

1.7.1 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.

1.7.2 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.

1.7.3 Architecture and protocol information

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

- *ARM architecture*
- *Interrupt controller 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 *ARM Architecture Reference Manual, ARMv7-A and ARMv7-R edition*.

Interrupt controller architecture

The Cortex-A9 MPCore processor implements Architecture version 1.0 of the ARM Generic Interrupt Controller architecture profile. See *ARM Generic Interrupt Controller Architecture Specification*.

Trace macrocell, optional

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

Advanced Microcontroller Bus Architecture

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

1.8 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*.

1.8.1 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.

1.8.2 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* 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 2-7
  - **MAXCLKLATENCY[2:0]** bus added. See *Configuration* on page A-5
  - Addition of CP15 power control register. See *Power Control Register* on page 4-125
- Extension of the Performance Monitoring Event bus. In r1p0, **PMUEVENT** is 52 bits wide:
  - Addition of Cortex-A9 specific events. See Table 4-72 on page 4-105.
  - Event descriptions extended. See Table 4-70 on page 4-102

- Addition of **PMUSECURE** and **PMUPRIV**. See *Performance monitoring signals* on page A-16.
- TLB options for 128 entries or 64 entries. See *TLB Type Register* on page 4-36.
- **DEFLAGS[6:0]** added. See *DEFLAGS[6:0]* on page 4-108
- 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 4-59.
- 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 C-10.
- ID Register values changed to reflect r1p0 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.

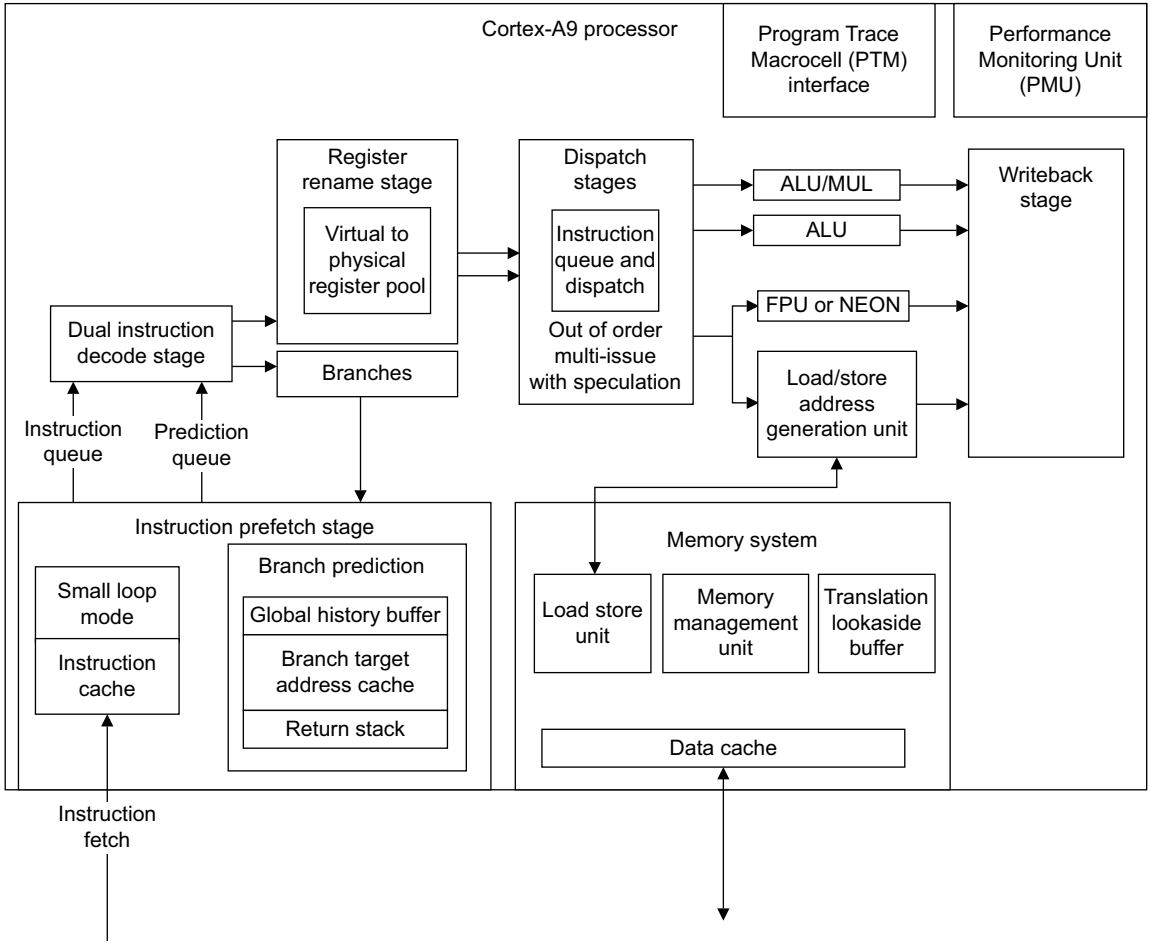


Figure 2-1 Cortex-A9 processor top-level diagram

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 2-11.

2.1.3 PTM interface

The Cortex-A9 processor implements 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 2-4.

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. See *Event Counter Selection Register* on page 4-100 and *Program Counter Sampling Register (DBGPCSR)* on page 8-24.

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 2-5 shows the PTM interface signals.

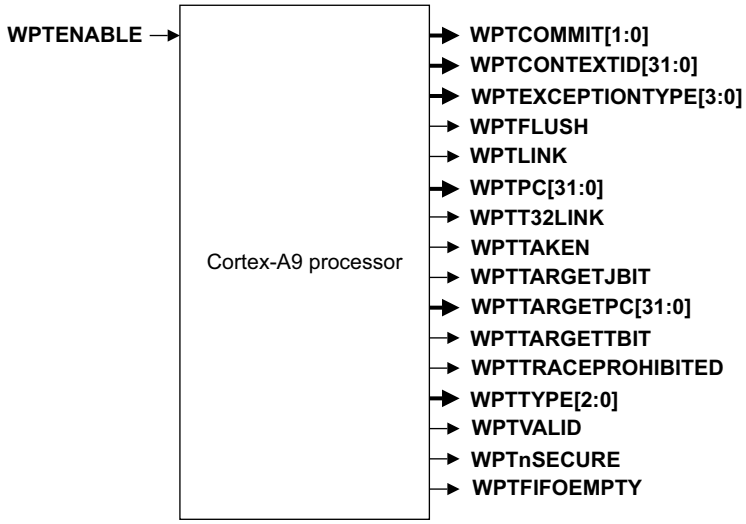


Figure 2-2 PTM interface signals

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**.

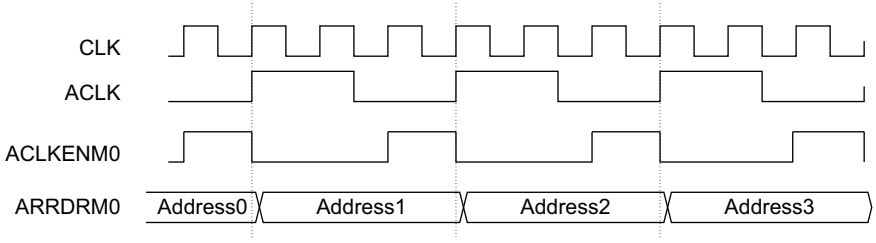


Figure 2-3 ACLKENM0 used with a 3:1 clock ratio

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 2-8.

2.4.1 Gated blocks

The CORTEXA9 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 Engineering 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:

- an enable bit
- the max clk latency bits.

See *Power Control Register* on page 4-125.

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 the 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, except for the debug logic and the Data Engine logic.

nDBGRESET The **nDBGRESET** signal is the reset that initializes the debug logic. See Chapter 8 *Debug*.

nDERESET The **nDERESET** signal is the reset that initializes the Data Engine logic.

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	nDERESET	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 **DECLKOFF**. See *Configuration* 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.

2.5.4 Date Engine logic reset

An additional reset controls the Data Engine independently of the Cortex-A9 processor reset. Use this reset to hold the Data Engine in a reset state so that the power to the Data Engine can be safely removed without placing any logic within the Data Engine unit in a different state

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 unwanted data RAMs
- small loop mode. If an instruction loop fits in four BTAC entries, then instruction cache accesses are turned off, 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 Data Engine
- a power domain for Cortex-A9 processor RAMs, apart from the Data Engine.

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 Data Engine disabled	Powered-up	Powered-up	Powered-up	See *Coprocessor Access Control Register* on page 4-65 for information about disabling the Data Engine.
		Clocked	No clock	
Run Mode with Data Engine powered off	Powered-up	Powered-up	Powered off	The Data Engine 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 2-13.
		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 Wait For Interrupt instruction.

Entry into WFE Mode is performed by executing the 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 2-15 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 **DECLAMP** 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 level shifters and 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-26.

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 power domains.

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

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.

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 Cortex-A9 implements the ARM architecture v7-A. This includes:

- the 32-bit ARM instruction set
- the Thumb instruction set, a variable-length instruction set, that has both 16-bit and 32-bit instructions
- the 8-bit Java bytecodes
- the ThumbEE instruction set architecture
- the security extensions, TrustZone
- the Advanced SIMD extensions, NEON.

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

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 4-128.

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 Processor operating states

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

ARM state 32-bit, word-aligned ARM instructions are executed in this state.

Thumb state 16-bit and 32-bit, halfword-aligned instructions.

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

Jazelle state Variable length, byte-aligned instructions.

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

Table 3-1 J and T bit encoding

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

Note

- Transition between ARM and Thumb states does not affect the processor mode or the register contents. For details on entering and exiting ThumbEE state, see the *ARM Architecture Reference Manual*.

3.4.1 Switching state

You can switch the operating state of the processor between:

- ARM state and Thumb state using the `BX` and `BLX` instructions, and loads to the PC. Switching state is described in the *ARM Architecture Reference Manual*.
- Thumb state and ThumbEE state using the `ENTERX` and `LEAVEX` instructions.

- ARM and Jazelle state using the BXJ instruction.
- Thumb and Jazelle state using the BXJ instruction.

Exceptions are entered, handled, and exited in ARM state or in Thumb state. See *System Control Register* on page 4-59 for details on how to select reset instruction state.

Exception return instructions restore the SPSR to the CPSR, which can also cause a transition back to any state.

3.4.2 Interworking ARM and Thumb code sequences

The processor enables you to freely mix ARM and Thumb code sequences. You can use BLX instructions to call ARM subroutines from Thumb.You can also use BLX instructions to call Thumb subroutines from ARM.

3.5 Data types

The Cortex-A9 processor supports the following data types:

Byte 8 bits
Halfword 16 bits
Word 32 bits
Doubleword 64 bits.

Note

- when any of these types are described as unsigned, the N-bit data value represents a non-negative integer in the range 0 to $+2^N-1$, using normal binary format
- when any of these types are described as signed, the N-bit data value represents an integer in the range -2^{N-1} to $+2^{N-1}-1$, using two's complement format.

For best performance you must align these in memory as follows:

- byte quantities can be placed on any byte boundary
- halfword quantities must align with 2-byte boundaries
- word quantities must align with 4-byte boundaries
- doubleword quantities must align with 8-byte boundaries.

The processor supports mixed-endian and unaligned accesses.

Note

You can only use `LDRD`, `LDM`, `LDC`, `STRD`, `STM`, or `STC` instructions to access word-aligned quantities.

3.6 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.

3.7 Addresses in the Cortex-A9 processor

Three distinct types of address exist in ARM processors:

- *Physical Address* (PA)
- *Virtual Address* (VA)
- *Modified Virtual Address* (MVA).

See the *ARM Architecture Reference Manual* for more details. 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 3-2 shows the address types in the processor system.

Table 3-2 Address types in the processor system

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

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

1. The Cortex-A9 processor issues the VA of the instruction as Secure or Non-secure VA according to the state the processor is in.
2. The instruction cache is indexed by the lower bits of the VA. The TLB performs the translation in parallel with the cache 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.

3.8 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*
- *Security extensions write access disable* on page 3-12.

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

3.8.1 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. See *Secure Configuration Register* on page 4-68.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. See *System control and configuration* on page 4-3.
6. Pass control through the Secure Monitor software to the non-secure OS with an `SMC` instruction to enable the Non-secure operating system to initialize. The required operations depend on the operating system, and typically include initialization of caches, MMU, exception vectors, and stacks.

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

3.8.2 Security extensions write access disable

The processor pin **CP15SDISABLE** disables write access to certain registers in the system control coprocessor. Attempts to write to these registers when **CP15SDISABLE** is HIGH result in an Undefined instruction exception. Reads from the registers are still permitted. For more information about the registers affected by this pin. See *System control and configuration* on page 4-3.

Chapter 4
The System Control Coprocessor

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

This section gives an overall view of the system control coprocessor. See *System control functional groups* for detail of the registers in 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.

4.1.1 System control functional groups

The system control coprocessor is a set of registers that you can write to and read from. The functional groups for the registers are:

- *System control and configuration* on page 4-3
- *MMU control and configuration* on page 4-4
- *Cache control and configuration* on page 4-4
- *System performance monitor registers* on page 4-11.

The system control coprocessor controls the operation of the security extensions:

- some of the registers are only accessible in the Secure state
- some of the registers are banked for Secure and Non-secure states
- some of the registers are common to Secure and Non-secure states.

Note

In Monitor mode, the Cortex-A9 processor is in Secure state. The processor treats all accesses as secure and the system control coprocessor behaves as if it operates in the Secure state regardless of the value of the NS bit, see *Secure Configuration Register* on page 4-68. In Monitor mode the NS bit defines the copies of the banked registers in the system control coprocessor that the processor can access:

NS = 0 Access to Secure state system registers.

NS = 1 Access to Non-secure state system registers.

Registers that are only accessible in the Secure state are always accessible in Monitor mode, regardless of the value of the NS bit.

4.1.2 System control and configuration

The purpose of the system control and configuration registers is to provide overall management of:

- security extensions behavior
- memory functionality
- interrupt behavior
- exception handling
- program flow prediction
- coprocessor access rights for control of NEON and FPU access rights.

The system control and configuration registers also provide the processor ID. Some of the functionality depends on how you set external signals at reset.

System control and configuration behaves in three ways:

- as a set of flags or enables for specific functionality
- as a set of numbers, values that indicate system functionality
- as a set of addresses for processes in memory.

TrustZone write access disable

The input pin, **CP15SDISABLE**, enables users to disable write access to some of the Secure registers. See Table 4-1 on page 4-4.

You can use the **CP15SDISABLE** pin to disable subsequent access to system control processor registers after the secure boot code runs. This protects the configuration set up by the Secure boot code.

A change to the **CP15SDISABLE** pin takes effect on the instructions decoded by the processor as quickly as possible. Software must perform an `ISB` after a change to this pin has occurred to ensure that its effects are recognized on following instructions. To ensure that this pin is effective you must do the following:

- control of the **CP15SDISABLE** pin remains within the SoC that implements the macrocell
- the **CP15SDISABLE** pin is set to logic 0 by the SoC hardware at reset.

When asserted HIGH, any attempt to write to the secure version of a banked register, NS-bit is 0, or any nonbanked register, NS-state is 0, results in an Undefined instruction exception.

Table 4-1 shows the system registers affected by the primary input pin, **CP15SDISABLE**.

Table 4-1 System registers affected by CP15SDISABLE

Register	Instruction
Control Register	`MCR p15, 0, <Rd>, c1, c0, 0`
Auxiliary Control Register	`MCR p15, 0, <Rd>, c1, c0, 1`
Translation Table Base 0	`MCR p15, 0, <Rd>, c2, c0, 0`
Translation Table Control Register	`MCR p15, 0, <Rd>, c2, c0, 2`
Domain Access Control	`MCR p15, 0, <Rd>, c3, c0, 0`
Primary Region Remap	`MCR p15, 0, <Rd>, c10, c2, 0`
Normal Memory Region Remap	`MCR p15, 0, <Rd>, c10, c2, 1`
Vector Base	`MCR p15, 0, <Rd>, c12, c0, 0`
Monitor Base	`MCR p15, 0, <Rd>, c12, c0, 1`
Configuration Base Address	`MCR p15,0,<Rd>,c15,c4,0`

4.1.3 MMU control and configuration

The purpose of the MMU control and configuration registers is to:

- generate physical address locations from the virtual addresses that the processor generates
- control program access to memory
- configure translation table memory type attributes
- provide information about MMU faults and external aborts
- translate and lock translation table walk entries
- hold thread and process IDs.

4.1.4 Cache control and configuration

The purpose of the cache control and configuration registers is to:

- provide information on the size and architecture of the instruction and data caches

- control cache maintenance operations that include clean and invalidate caches, drain and flush buffers, and address translation
- override cache behavior during debug or interruptible cache operations.

4.1.5 Cache Operations Registers

Cache operations registers manage the associated cache levels. The maintenance operations are in the following management groups:

- Set/Way:
  - clean
  - invalidate
  - clean and invalidate.
- VA:
  - clean
  - invalidate
  - clean and invalidate.

In addition, the maintenance operations use the following definitions:

Point of coherency (PoC)

The PoC is the point at which all agents that can access memory are guaranteed to see the same copy of a memory location

Point of unification (PoU)

The PoU is the point where the instruction and data caches, and the TLB translation table walks have merged for a uniprocessor system.

For Cortex-A9 processors the PoC and the PoU is at the L2 interfaces.

Note

- Reading from these registers, except for reads from the PA Register, causes an Undefined instruction exception.
- All accesses to these registers can only be executed in a privileged mode of operation, except Data Synchronization Barrier, Instruction Synchronization Barrier, and Data Memory Barrier. These can be executed in User mode. Attempting to execute a privileged instruction in User mode results in an Undefined instruction exception.

- For information on the behavior of the invalidate, clean, and prefetch operations in the Secure and Non-secure states, see the *ARM Architecture Reference Manual*.

Data formats for the cache operations

The possible formats for the data supplied to the cache maintenance and prefetch buffer operations depend on the specific operation:

- *Set/way*
- *VA* on page 4-7
- *SBZ* on page 4-7.

Set/way

Figure 4-1 shows the set/way bit assignments for invalidate and clean operations.

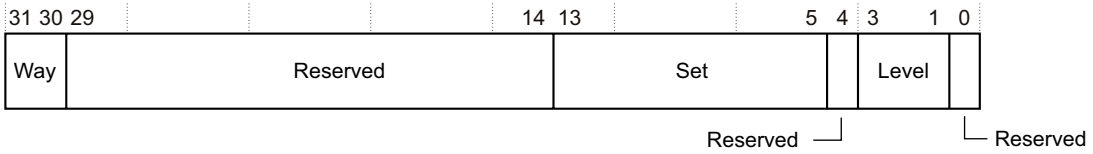


Figure 4-1 c7 set/way bit assignments

Table 4-2 shows how the bit values correspond with the Cache Operation functions for Set/Way format operations.

Table 4-2 Bit values for c7 set/way functions

Bits	Name	Description
[31:30]	Way	Selects the way for the c7 set/way cache operation: 2'b11: way3 2'b10: way2 2'b01: way1 2'b00: way0.
[29:14]	-	Reserved.
[13:5]	Set	Selects the set for the c7 Set/Way cache operation: 64KB cache uses bits[13:5] 32KB cache uses bits[12:5] 16KB cache uses bits[11:5].

Table 4-2 Bit values for c7 set/way functions (continued)

Bits	Name	Description
[4]	-	Reserved.
[3:1]	Level	SBZ.
[0]	-	SBZ.

See *Cache Size Selection Register* on page 4-58 for more information on cache sizes.

VA

Figure 4-2 shows the VA bit arrangement for invalidate, clean, and prefetch operations.

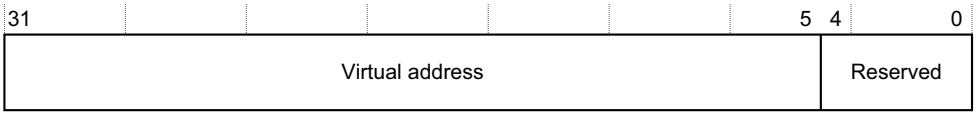


Figure 4-2 c7 VA bit assignments

Table 4-3 shows how the bit values correspond with the Cache Operation functions for VA operations.

Table 4-3 Bit values for VA functions

Bits	Name	Description
[31:5]	Virtual address	Specifies address to invalidate, clean, or prefetch
[4:0]	-	SBZ

SBZ

The value supplied Should Be Zero. The value `0x00000000` must be written to the register.

Table 4-4 shows the cache operation functions and the associated data and instruction formats for CP15 c7.

Table 4-4 Cache operation functions

Description	Mnemonic	Data	Instruction
Invalidate entire instruction cache Inner Shareable	ICIALLUIS	SBZ	`MCR p15, 0, <Rd>, c7, c1, 0`
Invalidate entire branch predictor array Inner Shareable	BPIALLIS	SBZ	`MCR p15, 0, <Rd>, c7, c1, 6`
Invalidate all instruction caches to PoU. Also flushes branch target cache.	ICIALLU	SBZ	`MCR p15, 0,<Rd>, c7, c5, 0`
Invalidate instruction cache by MVA to PoU	ICIMVAU	MVA	`MCR p15, 0,<Rd>, c7, c5, 1`
Invalidate entire branch predictor array	BPIALL	SBZ	`MCR p15, 0,<Rd>, c7, c5, 6`
Invalidate data cache line by MVA to PoC.	DCIMVAC	MVA	`MCR p15, 0,<Rd>, c7, c6, 1`
Invalidate data cache line by set/way.	DCISW	Set/Way	`MCR p15, 0,<Rd>, c7, c6, 2`
Clean data cache line to PoC by MVA	DCCMVAC	VA	`MCR p15, 0,<Rd>, c7, c10, 1`
Clean data cache line by set/way	DCCSW	Set/Way	`MCR p15, 0,<Rd>, c7, c10, 2`
Clean data or unified cache line by MVA to PoU.	DCCMVAU	MVA	`MCR p15, 0,<Rd>, c7, c11, 1`
Clean and invalidate data cache line by MVA to PoC.	DCCIMVAC	MVA	`MCR p15, 0,<Rd>, c7, c14, 1`
Clean and invalidate data cache line by set/way.	DCCISW	Set/Way	`MCR p15, 0,<Rd>, c7, c14, 2`

The cache invalidation operations apply to all cache locations. An explicit flush of the relevant lines in the branch target cache must be performed after invalidation of instruction cache lines or the results are Unpredictable. This is not required after an entire instruction cache invalidation.

The operations that act on a single cache line identify the line using the contents of Rd as the address, passed in the MCR instruction. The data is interpreted using:

- *Set/way format*
- *VA format* on page 4-10.

Set/way format

Figure 4-3 on page 4-9 shows the Set/Way format you can use to specify a line in the cache that must be accessed.

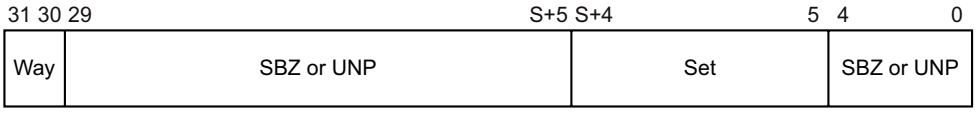


Figure 4-3 Set/Way bit assignments

Table 4-5 shows the bit fields for Set/Way operations using CP15 c7, and their meanings.

Table 4-5 Set/Way operations using CP15 c7 bit functions

Bits	Name	Description
[31:30]	Way	Way in set being accessed
[29:S+5]	-	SBZ or UNP
[S+4:5]	Set	Set being accessed
[4:0]	-	SBZ or UNP

The value of S in Table 4-6 is dependent on the cache size. Table 4-6 shows the relationship of cache sizes and the S parameter value.

Table 4-6 Cache size and S parameter dependency

Cache size	S parameter value
16KB	7
32KB	8
64KB	9

The value of S is derived from the following equation:

$$S = \log_2 \left(\frac{\text{Cache size}}{\text{Associativity x line length in bytes}} \right)$$

See *Cache Size Selection Register* on page 4-58 for details of instruction and data cache size.

Example 4-1 on page 4-10 shows the use of the command Clean Data Cache.

Example 4-1 Clean Data Cache

```
    ;code is specific to Cortex-A9 processors with 32KB caches
    MOV R0, #0:SHL:5
way_loop
    MOV R1, #0:SHL:30
line_loop
    ORR R2,R1,R0
    MCR p15,0,R2,c7,c10,2
    ADD R1,R1,#1:SHL:30
    CMP R1,#0
    BNE line_loop
    ADD R0,R0,#1:SHL:5
    CMP R0,#1:SHL:13
    BNE way_loop
```

VA format

The VA format is useful for flushing a particular address or range of addresses in the caches. Figure 4-4 shows the VA bit arrangement for c7 functions:

- Invalidate Instruction Cache Line
- Invalidate Data Cache Line
- Clean Data Cache Line
- Prefetch Instruction Cache Line
- Clean and Invalidate Data Cache Line.

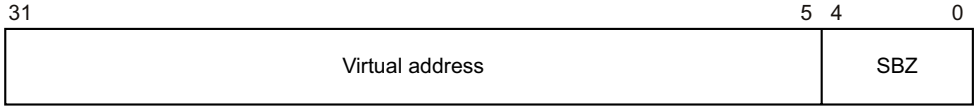


Figure 4-4 CP15 Register c7 VA bit assignments

Bits [4:0] are ignored.

All operations on entire data caches or instruction caches are interruptible and restartable. When interrupted, these operations stop and automatically restart from where they were interrupted.

User access to CP15 c7 operations

A small number of CP15 c7 operations can be executed by code while in User mode. Attempting to execute a privileged operation in User mode using CP15 c7 results in an Undefined instruction trap.

4.1.6 System performance monitor registers

The purpose of the performance monitor registers is to:

- control the monitoring operation
- count events.

System performance monitoring counts system events, such as cache misses, TLB misses, pipeline stalls, and other related features to enable system developers to profile the performance of their systems. *Performance Monitor Control Register* on page 4-92, *Program Counter Sampling Register (DBGPCSR)* on page 8-24 and *Performance monitoring signals* on page A-16 provide more information on system performance monitoring.

4.1.7 System feature registers

The system feature registers specify:

- Processor features
- Debug features
- Memory model features.

You can access these registers with the following CP15 instruction:

```
MRC p15,0,<Rd>,c0,c1,{0-7}  ; reads feature version registers
```

Depending on the Opcode_2 value, the accessed register is:

- Opcode_2 = 0 for ID_PFR0, Processor Feature Register 0
- Opcode_2 = 1 for ID_PFR1, Processor Feature Register 1
- Opcode_2 = 2 for ID_DFR0, Debug Feature Register 0
- Opcode_2 = 3 Reserved
- Opcode_2 = 4 for IDMMFR0, Memory Mode0 Feature Register 0
- Opcode_2 = 5 for IDMMFR1, Memory Model Feature Register 1
- Opcode_2 = 6 for IDMMFR2, Memory Mode2 Feature Register 2
- Opcode_2 = 7 for IDMMFR3, Memory Model3Feature Register 3.

The Reserved Opcode_2 value, Opcode_2=3, reads as zero.

4.1.8 c0, Instruction set attributes registers

The Instruction set attributes registers are:

- *Instruction Set Attributes Register 0* on page 4-48
- *Instruction Set Attributes Register 1* on page 4-49
- *Instruction Set Attributes Register 2* on page 4-51
- *Instruction Set Attributes Register 3* on page 4-52

- *Instruction Set Attributes Register 4* on page 4-53.

These registers are read-only registers and are accessed with the following CP15 instructions:

```
MRC p15,0,<Rd>,c0,c2,{0-7} ; reads feature version registers
```

Depending on the Opcode_2 value, the accessed register is:

- Opcode_2 = 0 for ID_ISAR0, ISA feature Register 0
- Opcode_2 = 1 for ID_ISAR1, ISA Feature Register 1
- Opcode_2 = 2 for ID_ISAR2, ISA Feature Register 2
- Opcode_2 = 3 for ID_ISAR3, ISA Feature Register 3
- Opcode_2 = 4 for ID_ISAR4, ISA Feature Register 4
- Opcode_2 = 5 Reserved
- Opcode_2 = 6 Reserved
- Opcode_2 = 7 Reserved.

4.1.9 c7, VA to PA operations

Writes to the VA to PA Translation Registers translate the virtual address provided by a general-purpose register (<Rn> Field) and store the corresponding physical address in the PA Register. Figure 4-5 shows the register bit arrangement.

Figure 4-5 VA to PA register bit assignments

The VA to PA translation can only be performed in privileged mode and uses the current ASID, in the Context ID Register, to perform the comparison in the TLB.

The VA to PA Translation Register is accessed by writing to CP15 c7 register with the <CRm> field set to c8 and the Opcode_2 field being used to select the kind of permission check that is performed during the translation.

With the processor in Non-secure state the operations are:

```
MCR p15,0,<Rn>,c7,c8,0; VA to PA translation with privileged read permission check
MCR p15,0,<Rn>,c7,c8,1; VA to PA translation with privileged write permission check
MCR p15,0,<Rn>,c7,c8,2; VA to PA translation with user read permission check
MCR p15,0,<Rn>,c7,c8,3; VA to PA translation with user write permission check.
```

With the processor in Secure state the operations are:

```
MCR p15,0,<Rn>,c7,c8,4; VA to PA translation with privileged read permission check
MCR p15,0,<Rn>,c7,c8,5; VA to PA translation with privileged write permission
                       ;check
MCR p15,0,<Rn>,c7,c8,6; VA to PA translation with user read permission check
MCR p15,0,<Rn>,c7,c8,7; VA to PA translation with user write permission check.
```

4.1.10 c8, TLB Operations Register

The purpose of the TLB Operations Register is to either:

- invalidate all the unlocked entries in the TLB
- invalidate all TLB entries for an area of memory before the OS remaps it
- invalidate all TLB entries that match an ASID value.

The TLB Operations Register is:

- in CP15 c8
- a 32-bit write-only register
- accessible in privileged modes only.

The TLB Operations Register, CP15 c8, is a write-only register used to manage the TLB.

Table 4-7 shows the defined TLB operations. Select the function to be performed using the Opcode_2 and CRm fields in the `MCR` instruction used to write CP15 c8. Writing other Opcode_2 or CRm values is Unpredictable.

Reading from CP15 c8 is Undefined.

Table 4-7 shows the TLB Operations Register instructions.

Table 4-7 TLB Operations Register instructions

Description	Mnemonic	Data	Instruction
Invalidate entire Unified TLB Inner Shareable	TLBIALLIS	SBZ	`MCR p15,0,<Rd>,c8,c3,0`
Invalidate Unified TLB entry by VA Inner Shareable	TLBIMVAIS	VA/ASID	`MCR p15,0,<Rd>,c8,c3,1`
Invalidate Unified TLB entry by ASID match Inner Shareable	TLBIASIDIS	ASID	`MCR p15,0,<Rd>,c8,c3,2`
Invalidate Unified TLB entry by VA all ASID Inner Shareable	TLBIMVAAIS	VA	`MCR p15,0,<Rd>,c8,c3,3`
Invalidate entire Unified TLB	TLBIALL	Ignored	`MCR p15,0,<Rd>,c8,c7,0`

Table 4-7 TLB Operations Register instructions (continued)

Description	Mnemonic	Data	Instruction
Invalidate Unified TLB by VA	TLBIMVA	VA	`MCR p15,0,<Rd>,c8,c7,1`
Invalidate TLB entries by ASID Match	TLBIASID	ASID	`MCR p15,0,<Rd>,c8,c7,2`
Invalidate TLB entries by VA All ASID	TLBIMVAA	VA/ASID	`MCR p15, 0, <Rd>, c8, c7,3`

The CRm value indicates to the hardware what type of access caused the TLB function to be invoked.

Table 4-8 shows the CRm values for the TLB Operations Register, and their meanings. All other CRm values are reserved.

Table 4-8 CRm values for TLB Operations Register

CRm	Meaning
c3	Inner Shareable broadcast operations. Used with Cortex-A9 Multiprocessors[a]
c5	Instruction TLB operation
c6	Data TLB operation
c7	Unified TLB operation

a. See *Auxiliary Control Register* on page 4-63 for information on broadcast operations.

Note

The Cortex-A9 processor has a unified TLB. Any TLB operations specified for the instruction or data micro TLBs perform the equivalent operation on the unified TLB.

Figure 4-6 shows the bit arrangement of the TLB operations that use the Virtual Address as an argument.

Figure 4-6 TLB Operations Register Virtual Address bit assignments

The Invalidate TLB Entries on ASID Match function requires an ASID as an argument. Figure 4-7 on page 4-15 shows the bit arrangement of this.

Figure 4-7 TLB Operations Register ASID bit assignments

Functions that update the contents of the TLB occur in program order. Therefore, an explicit data access prior to the TLB function uses the old TLB contents, and an explicit data access after the TLB function uses the new TLB contents. For instruction accesses, TLB updates are guaranteed to have taken effect before the next pipeline flush. This includes ISB operations and exception return sequences.

Invalidate TLB Entries on ASID Match

This is a single interruptible operation that invalidates all TLB entries that match the provided 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.

Invalidate TLB entry by VA all ASID

You can use Invalidate TLB Entries to invalidate an area of memory prior to remapping. You must perform an Invalidate TLB entry by VA all ASID in each area to be remapped as either section, small page, or large page.

This function invalidates an unlocked TLB entry that matches the provided VA. This entry can be global or nonglobal. If the entry is nonglobal the ASID matching is ignored. This function does not invalidate a matching locked entry.

4.1.11 c10, Memory region remap

The remap capability falls into two levels, the primary remap, enables the primary memory type (Normal, Device, or Strongly ordered) to be remapped. For Device and Normal memory, the effect of the S bit can be independently remapped. To provide maximum flexibility, this level of remapping enables regions that were originally not Normal memory to be remapped independently

The remapping is applied to all sources of TLB requests.

After this primary remapping is performed any region that is mapped as Normal memory can have the inner and outer cacheable attributes determined by the Normal memory Remap register.

The memory region remap registers are accessed by:

```
MCR/MRC{cond} p15, 0, Rd, c10, c2, 0;access primary memory region remap register
```

```
MCR/MRC{cond} p15, 0, Rd, c10, c2, 1;access normal memory region remap register
```

These registers are used to remap memory region types. This remapping is enabled when the TRE bit of the System Control Register (SCTLR) is set. The remapping takes place on the page table values, and overrides the settings specified in the page tables,. The remapping does not take place when the MMU is turned off.

Table 4-9 and Table 4-10 show the encoding used for each memory type.

The primary region remap register determines the memory type and also the treatment of the shareable attribute, and the subsequent normal memory remap register applies to memory regions that are mapped as normal memory. This normal memory region remap register enables definition of the inner cacheable and outer cacheable attributes.

Table 4-9 shows the primary remapping encodings.

Table 4-9 Primary remapping encodings

Region	Encoding
Strongly-ordered	00
Shared Device	01
Normal Memory	10
Unpredictable	11

Table 4-10 shows the normal remapping encodings.

Table 4-10 Inner or outer region type encodings

Inner or Outer Region	Encoding
Noncacheable	00
Write-Back, Write-Allocate	01
Write-Through, no-Write-Allocate	10
Write-Back, no Write-Allocate	11

4.1.12 c13, Software Thread ID registers

The purpose of the thread and process ID registers is to provide locations to store the IDs of software threads and processes for OS management purposes. These registers have no effect on processor behavior.

The thread and process ID registers are:

- in CP15 c13
- three 32-bit read and write registers
  - — User read and write Thread and Process ID Register, TPIDRURW
    Read and write in User and privileged modes
  - — User Read Only Thread and Process ID Register, TPIDRURO
    Read only in User mode, read and write in privileged modes
  - — Privileged Only Thread and Process ID Register, TPIDRPRW.
    Read and write in privileged modes only.

You can access the thread registers by reading or writing to CP15 c13 with the Opcode_2 field set to 2, 3 or 4:

```
MRC p15,0,<Rd>,c13,c0,2/3/4; Read Thread ID registers
MCR p15,0,<Rd>,c13,c0,2/3/4; Write Thread ID registers
```

Figure 4-8 shows the Thread ID registers bit assignment.

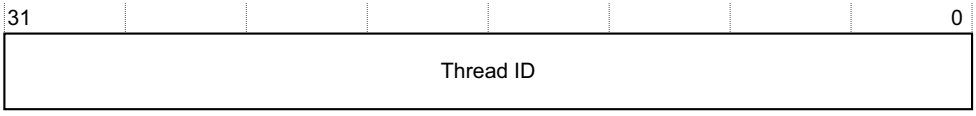


Figure 4-8 Thread ID registers bit assignments

Thread ID registers have different access rights depending on Opcode_2 field value:

- Opcode_2 = 2: This register is both user and privileged RW accessible.
- Opcode_2 = 3: This register is user read-only and privileged RW accessible.
- Opcode_2 = 4: This register is privileged RW accessible only.

4.1.13 c15, TLB lockdown operations

TLB lockdown operations enable saving or restoring lockdown entries in the TLB. Table 4-11 shows the defined TLB lockdown operations.

Table 4-11 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 4-9 shows the bit assignment of the index register used to access the lockdown TLB entries.

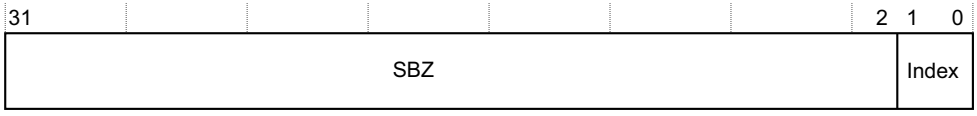


Figure 4-9 Lockdown TLB index bit assignments

Figure 4-10 shows the bit arrangement of the TLB VA Register format.

Figure 4-10 TLB VA Register bit assignments

Table 4-12 the TLB VA Register bit assignments.

Table 4-12 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]	-	Reserved (SBZ).
[10]	NS	NS bit.
[9:0]	Process	Memory space identifier.

Figure 4-11 shows the bit arrangement of the memory space identifier.

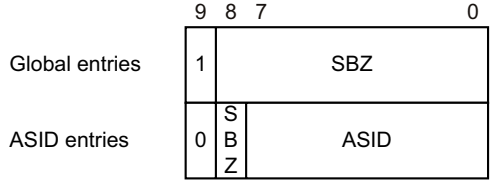


Figure 4-11 Memory space identifier format

Figure 4-12 shows the TLB PA Register bit assignment.

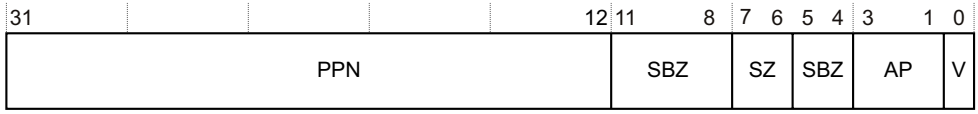


Figure 4-12 TLB PA Register bit assignments

Table 4-13 describes the functions of the TLB PA Register bits.

Table 4-13 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 SBZ when written.
[11:8]	-	Reserved. SBZ.
[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]	-	Reserved. SBZ
[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 4-13 shows the bit assignments of the TLB Attributes Register.

Figure 4-13 Main TLB Attributes Register bit assignments

Table 4-14 shows the TLB Attributes Register bit assignments. The Cortex-A9 processor does not support subpages.

Table 4-14 TLB Attributes Register bit assignments

Bits	Name	Description
[31:12]	-	SBZ.
[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.

4.1.14 Debug

The purpose of the debug registers is to support programmers when debugging their software on the processor. You can use the debug registers to:

- Communicate with an external host
- Configure debug hardware such as breakpoint and watchpoint functionality in the processor.

See Chapter 8 *Debug* for a description of the debug registers accessed through CP14.

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)
  - Lock (L).
- Reset is the reset value of the register

All system control coprocessor registers are 32 bits wide. Reserved register addresses are RAZ/WI.

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 Virtualization

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.

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.

4.2.3 c0 summary table

Table 4-15 shows the system control registers when CRm is c0.

Table 4-15 c0 system control registers

Op1	CRm	Op2	Name	Type	Reset	Description
0	c0	0	MIDR	RO	0x411FC090	*Main ID Register* on page 4-33
		1	CTR	RO	0x83338003	*Cache Type Register* on page 4-34
		2	TCMTR	RO	0x00000000	*TCM Type Register* on page 4-35
		3	TLBTR[a]	RO	-	*TLB Type Register* on page 4-36
		4	-	-	-	-
		5	MPIDR	RO	-	*Multiprocessor Affinity Register* on page 4-37
		6-7	-		-	-
	c1	0	ID_PFR0	RO	0x00001231	*Processor Feature Register 0* on page 4-39
		1	ID_PFR1	RO	0x00000011	*Processor Feature Register 1* on page 4-40
		2	ID_DFR0	RO	0x00010444	*Debug Feature Register 0* on page 4-41
		3	Reserved	-	-	-
		4	ID_MMFR0	RO	0x00100103	*Memory Model Features Register 0* on page 4-42
		5	ID_MMFR1	RO	0x20000000	*Memory Model Features Register 1* on page 4-44
		6	ID_MMFR2	RO	0x01230000	*Memory Model Features Register 2* on page 4-45
		7	ID_MMFR3	RO	0x00102111	*Memory Model Features Register 3* on page 4-47

Table 4-15 c0 system control registers (continued)

Op1	CRm	Op2	Name	Type	Reset	Description
0	c2	0	ID_ISAR0	RO	0x00101111	*Instruction Set Attributes Register 0* on page 4-48
		1	ID_ISAR1	RO	0x13112111	*Instruction Set Attributes Register 1* on page 4-49
		2	ID_ISAR2	RO	0x21232041	*Instruction Set Attributes Register 2* on page 4-51
		3	ID_ISAR3	RO	0x11112131	*Instruction Set Attributes Register 3* on page 4-52
		4	ID_ISAR4	RO	0x00011142	*Instruction Set Attributes Register 4* on page 4-53
		5-7	Reserved	-	-	-
	c3-c7	0-7	Reserved	-	-	-
1	c0	0	CCSIDR	RO	-	*Cache Size Identification Register* on page 4-54
		1	CLIDR	RO	0x09000003	*Cache Level ID Register* on page 4-56
		7	AIDR	RO	0x00000000	*Auxiliary ID Register* on page 4-57
2	c0	0	CSSELR	RW	-	*Cache Size Selection Register* on page 4-58
3-7	c0-c15	0-7	-		-	-

a. Depends on TLBSIZE. See *TLB Type Register* on page 4-36

4.2.4 c1 summary table

Table 4-16 shows the system control registers when CRn is c1.

Table 4-16 c1 system control registers

Op1	CRm	Op2	Name	Type	Reset	Description
0	c0	0	SCTLR	RW	0x00C50078	*System Control Register* on page 4-59
		1	ACTLR[a]	RW	0x00000000	*Auxiliary Control Register* on page 4-63
		2	CPACR	RW	[b]	*Coprocessor Access Control Register* on page 4-65
	c1	0	SCR[c]	RW	0x00000000	*Secure Configuration Register* on page 4-68
		1	SDER[c]	RW	0x00000000	*Secure Debug Enable Register* on page 4-70
		2	NSACR	RW[d]	-[e]	*Non-secure Access Control Register* on page 4-71
		3	VCR[c]	RW	0x00000000	*Virtualization Control Register* on page 4-73

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. See *Virtualization* on page 4-22.
d. This is a read and write register in Secure state and a read-only register in the Non-secure state.
e. 0x00000000 if NEON present and 0x0000C000 if NEON not present

4.2.5 c2 summary table

Table 4-17 shows the system control registers when CRn is c2.

Table 4-17 c2 system control registers

Op1	CRm	Op2	Name	Type	Reset	Description
0	c0	0	TTBR0	RW	-	*Translation Table Base Register 0* on page 4-75
		1	TTBR1	RW	-	*Translation Table Base Register 1* on page 4-77
		2	TTBCR	RW	0x00000000[a]	*Translation Table Base Control Register* on page 4-79

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

4.2.6 c3 summary table

Table 4-18shows the system control register when CRn is c3.

Table 4-18 c3 system control register

Op1	CRm	Op2	Name	Type	Reset	Description
0	c0	0	DACR	RW	-	*Domain Access Control Register* on page 4-82

4.2.7 c4, c5, and c6 summary tables

There are no operations where CRn is c4.

Table 4-19 shows the system control registers when CRn is c5.

Table 4-19 c5 system control registers

Op1	CRm	Op2	Name	Type	Reset	Description
0	c0	0	DFSR	RW	-	*Data Fault Status Register* on page 4-83
		1	IFSR	RW	-	*Instruction Fault Status Register* on page 4-86
	c1	0	ADFSR	-	-	*Auxiliary Data Fault Status Register* on page 4-88
		1	AIFSR	-	-	*Auxiliary Instruction Fault Status Register* on page 4-88

Table 4-20 shows the system control registers when CRn is c6.

Table 4-20 c6 system control registers

Op1	CRm	Op2	Name	Type	Reset	Page
0	c0	0	DFAR	RW	-	*Data Fault Address Register* on page 4-89
		2	IFAR	RW	-	*Instruction Fault Address Register* on page 4-89

4.2.8 c7 summary table

Table 4-21 shows the system control registers when CRn is c7.

Table 4-21 c7 system control registers

Op1	CRm	Op2	Name	Type	Reset	Description
0	c0	0-3	Reserved	WO	-	-
		4	NOP[a]	WO	-	*NOP Register* on page 4-90
	c1	0	ICIALLUIS	WO	-	*Cache Operations Registers* on page 4-5
		6	BPIALLIS	WO	-	
		7	Reserved	WO	-	-
	c4	0	PAR	RW	-	*Physical Address Register* on page 4-90
	c5	0	ICIALLU	WO	-	*Cache Operations Registers* on page 4-5
		1	ICIMVAU	WO	-	
		2-3	Reserved	WO	-	-
		4	ISB	WO	User	*Deprecated registers* on page 4-22
		5	-	-	-	-
		6	BPIALL	WO	-	*Cache Operations Registers* on page 4-5
	c6	0	-	-	-	-
		1	DCIMVAC	WO	-	*Cache Operations Registers* on page 4-5
		2	DCISW	WO	-	
	c8	0-7	V2PCWPR	WO	-	*c7, VA to PA operations* on page 4-12

Table 4-21 c7 system control registers (continued)

Op1	CRm	Op2	Name	Type	Reset	Description
0	c10	0	-	-	-	-
		1	DCCVAC	WO	-	*Cache Operations Registers* on page 4-5
		2	DCCSW	WO	-	*Cache Operations Registers* on page 4-5
		3	-	WO	-	-
		4	DSB	WO	User	*Deprecated registers* on page 4-22
		5	DMB	WO	User	
		6-7	-	-	-	-
	c11	0	-	-	-	-
		1	DCCVAU	WO	-	*Cache Operations Registers* on page 4-5
	c14	0	-	-	-	-
		1	DCCIMVAC	WO	-	*Cache Operations Registers* on page 4-5
		2	DCCISW	WO	-	*Cache Operations Registers* on page 4-5

a. This operation is performed by the `WFI` instruction. See also *Deprecated registers* on page 4-22.

4.2.9 c8 summary table

c8, TLB Operations Register on page 4-13 describes the operations where CRn is c8. *TLB Lockdown Register* on page 4-113 describes the implementation defined TLB lockdown operation.

4.2.10 c9 summary table

Table 4-22 shows the system control registers when CRn is c9.

Table 4-22 c9 system control registers

Op1	CRm	Op2	Name	Type	Reset	Description
0	c12	0	PMCR	RW	0x41093000	*Performance Monitor Control Register* on page 4-92
		1	PMCNTENSET	RW	0x00000000	*Count Enable Set Register* on page 4-94
		2	PMCNTENCLR	RW	0x00000000	*Count Enable Clear Register* on page 4-96
		3	PMOVSR	RW	-	*Overflow Flag Status Register* on page 4-97
		4	PMSWINC	WO	-	*Software Increment Register* on page 4-98
		5	PMSELR	RW	0x00000000	*Event Counter Selection Register* on page 4-100
	c13	0	PMCCNTR	RW	-	*Cycle Count Register* on page 4-101
		1	PMXEVTYPER	RW	-	*Event Selection Register* on page 4-101
		2	PMXEVCNTR	RW	-	*Performance Monitor Count Registers* on page 4-109
	c14	0	PMUSERENR	RW[a]	0x00000000	*User Enable Register* on page 4-110
		1	PMINTENSET	RW	0x00000000	*Interrupt Enable Set Register* on page 4-111
		2	PMINTENCLR	RW	0x00000000	*Interrupt Enable Clear Register* on page 4-112

a. RO in user mode

4.2.11 c10 summary table

Table 4-23 shows the system control registers when CRn is c10.

Table 4-23 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 4-113
	c2	0	PRRR	RW	0x00098AA4	*Primary Region Remap Register* on page 4-115
		1	NRRR	RW	0x44E048E0	*Normal Memory Remap Register* on page 4-116

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

4.2.12 c11 summary table

There are no system control registers where CRn is c11.

4.2.13 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* on page 4-118
		1	MVBAR	RW	-	*Monitor Vector Base Address Register* on page 4-120
	c1	0	Interrupt Status Register	RO	0x00000000	*Interrupt Status Register* on page 4-121
		1	Virtualization Interrupt Register	RO[b]	0x00000000	*Virtualization Interrupt Register* on page 4-122

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.

4.2.14 c13 summary table

Table 4-25 shows the system control registers when CRn is c13.

Table 4-25 c13 system control registers

Op1	CRm	Op2	Name	Type	Reset	Description
0	c0	0	-	RW	0x00000000	*Deprecated registers* on page 4-22
		1	Context ID	RW	-	*Context ID Register* on page 4-124
		2	TPIDRURW	RW[a]	-	*c13, Software Thread ID registers* on page 4-17
		3	TPIDRURO	RO[b]	-	
		4	TPIDRPRW.	RW	-	

a. RW in User mode
b. RO in User mode

4.2.15 c15 summary table

Table 4-26 shows the system control registers when CRn is c15.

Table 4-26 c15 system control registers

Op1	CRm	Op2	Name	Type	Reset	Description
0	c0	0	Power Control Register	RW[a]	[b]	*Power Control Register* on page 4-125
4	c0	0	Configuration Base Address	RO[c]	[d]	*Configuration Base Address Register* on page 4-126
5	c4	2	Read main TLB entry register	WO[e]	-	*c8, TLB Operations Register* on page 4-13
		4	Write main TLB entry register	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* on page A-5.
c. RW in Secure privileged mode and RO in Non-secure state and user secure state.

d. In Cortex-A9 processor 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.

4.3 System control coprocessor register descriptions

This section describes the system registers in the order in which they appear in the summary tables.

4.3.1 Main ID Register

The MIDR characteristics are:

Purpose Returns the device ID code that contains information about the processor.

Usage constraints The MIDR 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 4-15 on page 4-23.

Figure 4-14 shows the MIDR bit assignments.

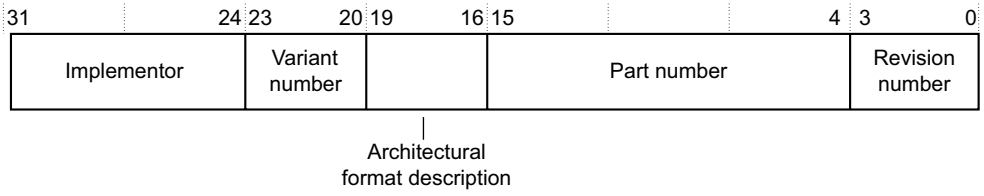


Figure 4-14 MIDR bit assignments

Table 4-27 shows the MIDR bit assignments.

Table 4-27 MIDR bit assignments

Bits	Value	Description
[31:24]	-	Implementor
[23:20]	-	Variant number In ARM implementations this is the major revision number n of the r*n*p*n* revision status.

Table 4-27 MIDR bit assignments (continued)

Bits	Value	Description
[19:16]	-	Architectural format description
[15:4]	-	Part number
[3:0]	-	Revision In ARM implementations this is the minor revision number n of the r*n*p*n* revision status.

To access the MIDR, use:

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

4.3.2 Cache Type Register

The CTR characteristics are:

Purpose Provides information about the size and architecture of the cache for the operating system.

Usage constraints The CTR 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 4-15 on page 4-23.

Figure 4-15 shows the CTR bit assignments.

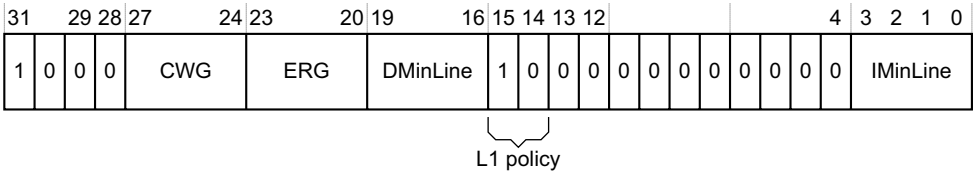


Figure 4-15 CTR bit assignments

Table 4-28 shows the CTR register bit assignments.

Table 4-28 CTR bit assignments

Bits	Name	Description
[31-29]	-	ARMv7 register format.
[28]	-	Read-As-Zero.
[27:24]	CWG	Cache Writeback Granule. See *ARM Architecture Reference Manual, ARMv7-A and ARMv7-R edition*
[23:20]	ERG	Exclusives Reservation Granule. See *ARM Architecture Reference Manual, ARMv7-A and ARMv7-R edition*
[19:16]	DMinLine	Log_2 of the number of words in the smallest cache line of all the data and unified caches under the core control.
[15:14]	L1policy	Indicates the level 1 instruction cache policy for indexing and tagging. b10 virtual index, physical tag, VIPT.
[13:4]	-	Read-As-Zero
[3:0]	IMinLine	Log_2 of the number of words in the smallest cache line of all the instruction caches under the control of the processor.

To access the CTR, use:

```
MRC p15,0,<Rd>,c0,c0,1; returns cache details
```

4.3.3 TCM Type Register

The TCMTR characteristics are:

Purpose Specifies that the processor does not implement instruction and data TCMs.

Usage constraints The TCMTR 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 4-15 on page 4-23.

To access the TCMTR, use:

```
MRC p15, 0, <Rd>, c0, c0, 2; Read TCM Type Register
```

4.3.4 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 4-15 on page 4-23.

Figure 4-16 shows the TLBTR bit assignments.

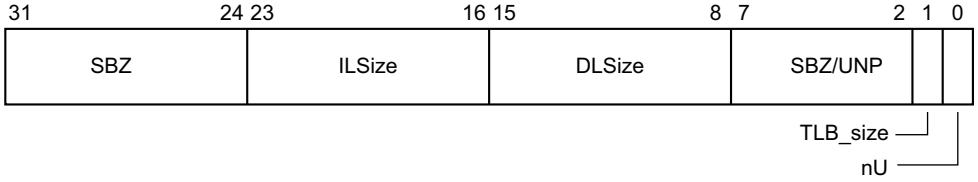


Figure 4-16 TLBTR bit assignments

Table 4-29 shows the TLBTR bit assignments.

Table 4-29 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.

Table 4-29 TLBTR bit assignments (continued)

Bits	Name	Description
[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
```

4.3.5 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 4-15 on page 4-23.

Figure 4-17 on page 4-38 shows the MPIDR bit assignments.

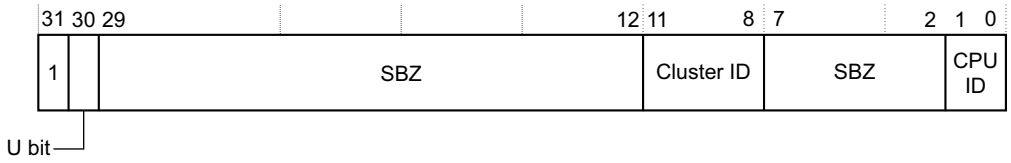


Figure 4-17 MPIDR bit assignments

Table 4-30 shows the MPIDR bit assignments.

Table 4-30 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
```

4.3.6 Processor Feature Register 0

The ID_PFR0 characteristics are:

Purpose Provides:

- information about the programmers model.
- top-level information about the instruction set support for the processor

Usage constraints Must be interpreted with ID_PFR1.

Configurations Available in all configurations.

Attributes See the register summary in Table 4-15 on page 4-23.

Figure 4-18 shows the ID_PFR0 bit assignments.

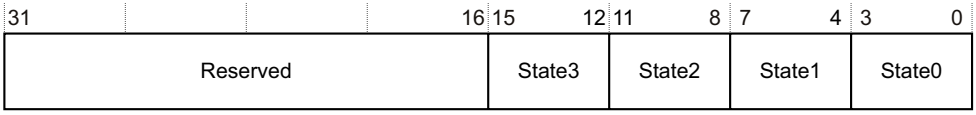


Figure 4-18 ID_PFR0 bit assignments

Table 4-31 shows the ID_PFR0 bit assignments.

Table 4-31 ID_PFR0 bit assignments

Bits	Name	Description
[31:16]	Reserved	RAZ.
[15:12]	State3	`0x1` ThumbEE instruction set supported.
[11:8]	State2	`0x2` Jazelle extension interface supported with clearing of JOSCR.CV on exception entry.
[7:4]	State1	`0x3` Support for Thumb encoding after the introduction of Thumb-2 technology, and for all 16-bit and 32-bit Thumb basic instructions.
[3:0]	State 0	`0x1` 32-bit ARM instruction set supported.

To access ID_PFR0, use:

```
MRC p15,0,<Rd>,c0,c1,0
```

4.3.7 Processor Feature Register 1

The ID_PFR1 characteristics are:

Purpose Provides information about the execution state support and programmers model for the processor.

Usage constraints The ID_PFR1 is:

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

The ID_PFR1 must be interpreted with ID_PFR0.

Configurations Available in all configurations.

Attributes See the register summary in Table 4-15 on page 4-23.

Figure 4-19 shows the ID_PFR1 Register bit assignments.

Figure 4-19 ID_PFR1 bit assignments

Table 4-32 shows the ID_PFR1 bit assignments.

Table 4-32 ID_PFR1 bit assignments

Bits	Name	Description
[31:12]	Reserved	RAZ.
[11:8]	Microcontroller programmers model	Microcontroller programmers model, not supported. `0x0` Processor does not support the microcontroller programmers model.
[7:4]	Security extension	`0x1` Security extension architecture v1 supported.
[3:0]	Programmers model	`0x1` standard ARMv4 programmers model and later.

To access ID_PFR1, use:

```
MRC p15,0,<Rd>,c0,c1,1
```

4.3.8 Debug Feature Register 0

The ID_DFR0 characteristics are:

Purpose Provides information about the debug system for the processor.

Usage constraints The ID_DFR0 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 4-15 on page 4-23.

Figure 4-20 shows the ID_DFR0 bit assignments.

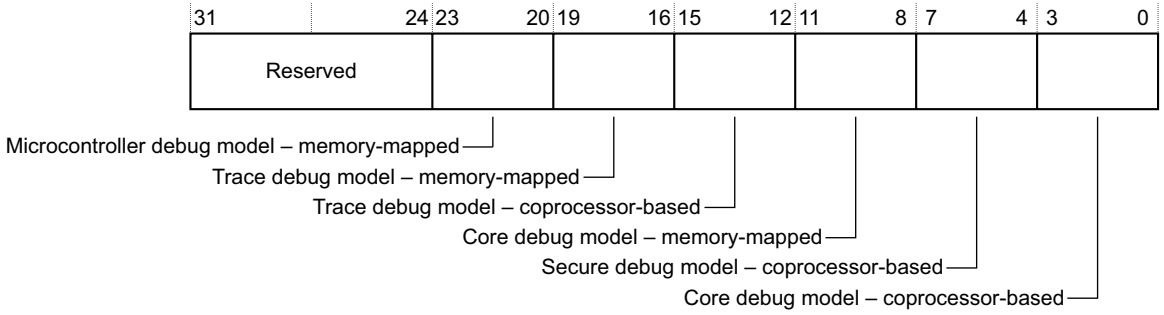


Figure 4-20 ID_DFR0 bit assignments

Table 4-33 shows the ID_DFR0 bit assignments.

Table 4-33 ID_DFR0 bit assignments

Bits	Name	Description
[31:24]	Reserved	RAZ.
[23:20]	Microcontroller debug model – memory-mapped	Indicates support for the microcontroller debug model. The Cortex-A9 processor does not support the microcontroller debug model – memory-mapped.
[19:16]	Trace debug model – memory-mapped	Indicates support for the trace debug model – memory-mapped. The Cortex-A9 processor supports the trace debug model – memory-mapped.

Table 4-33 ID_DFR0 bit assignments (continued)

Bits	Name	Description
[15:12]	Trace debug model – coprocessor-based	Indicates support for the coprocessor-based trace debug model. The Cortex-A9 processor does not support the trace debug model – coprocessor.
[11:8]	Core debug model – memory mapped	Indicates support for the memory-mapped debug model. The Cortex-A9 processor supports the memory-mapped debug model.
[7:4]	Secure Debug Model– coprocessor-based	v7model using cp14.
[3:0]	Core Debug Model– coprocessor-based	v7model using cp14.

To access ID_DFR0, use:

```
MRC p15,0,<Rd>,c0,c1,2
```

4.3.9 Auxiliary Feature Register 0

The ID_AFR0 characteristics are:

Purpose Can provide additional information about the features of the processor. Not used in this implementation.

Table 4-34 shows the ID_AFR0 bit assignments.

Table 4-34 ID_AFR0 bit assignments

Bits	Name	Description
[31:0]	-	RAZ/WI

To access the ID_AFR0, use:

```
MRC p15, 0, <Rd>, c0, c1, 3 ; Read Auxiliary Feature Register 0
```

4.3.10 Memory Model Features Register 0

The ID_MMFR0 characteristics are:

Purpose Provides information about the memory model, memory management, cache support, and TLB operations of the processor.

Usage constraints The ID_MMFR0 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 4-15 on page 4-23.

Figure 4-21 shows the ID_MMFR0 bit assignments.

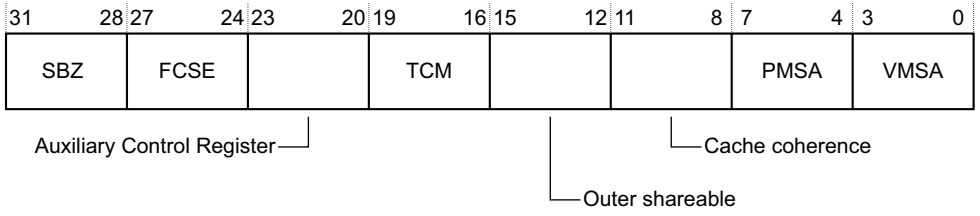


Figure 4-21 ID_MMFR0 bit assignments

Table 4-35 shows the ID_MMFR0 bit assignments.

Table 4-35 ID_MMFR0 bit assignments

Bits	Name	Description
[31:28]	-	SBZ
[27:24]	FCSE	FCSE not supported
[23:20]	Auxiliary Control Register	Auxiliary Control Register only
[19:16]	TCM and associated DMA support	Not supported
[15:12]	Outer Shareable	Outer Shareable not supported
[11:8]	Cache coherence support	L1 cache Shared support for Cortex-A9 MPCore processors
[7:4]	PMSA support	Not supported
[3:0]	VMSA support	VMSAv7with support for remapping and the access flag. ARMv7-A profile

To access the ID_MMFR0, use:

```
MRC p15, 0,<Rd>, c0, c1, 4
```

4.3.11 Memory Model Features Register 1

The ID_MMFR1 characteristics are:

Purpose Provides information about the memory model, memory management, cache support, and TLB operations of the processor.

Usage constraints The ID_MMFR1 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 4-15 on page 4-23.

Figure 4-22 shows the ID_MMFR1 bit assignments.

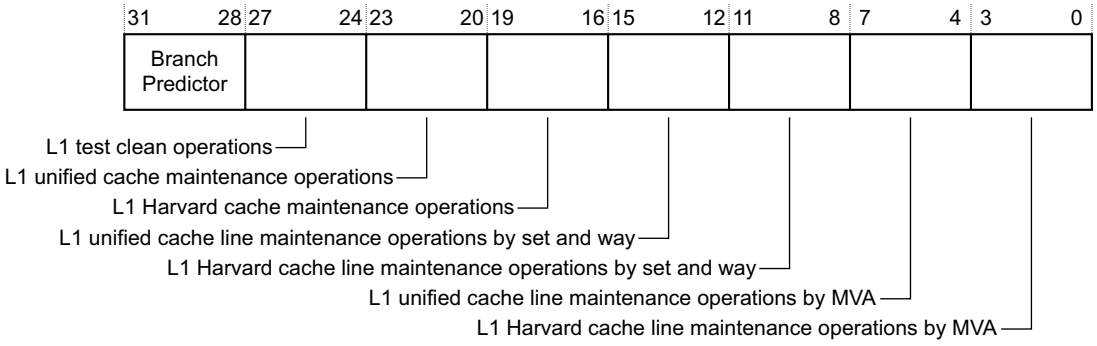


Figure 4-22 ID_MMFR1 bit assignments

Table 4-36 shows the ID_MMFR1 bit assignments.

Table 4-36 ID_MMFR1 bit assignments

Bits	Name	Description
[31:28]	Branch Predictor	Requires flushing on: • enabling or disabling the MMU • writing new data to instruction locations • writing new mappings to the translation tables • any change to the TTBR0, TTBR1, or TTBCR registers without a corresponding change to the FCSE ProcessID or ContextID.
[27:24]	L1 test and clean operation on data cache, Harvard or unified	Not supported.
[23:20]	L1 cache, all, maintenance operation, unified	Not supported.
[19:16]	L1 cache (all) maintenance operation, Harvard	Not supported.
[15:12]	L1 cache line maintenance operation by Set Way combination, unified	Not supported.
[11:8]	L1 cache line maintenance operation by Set Way combination, Harvard	Not supported.
[7:4]	L1 cache line maintenance operation by MVA, unified	Not supported.
[3:0]	L1 cache line maintenance operation by MVA, Harvard	Not supported.

To access the ID_MMFR1 use:

```
MRC p15, 0,<Rd>, c0, c1, 5
```

4.3.12 Memory Model Features Register 2

The ID_MMFR2 characteristics are:

Purpose Provides information about the memory model, memory management, cache support, and TLB operations of the processor.

Usage constraints The ID_MMFR2 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 4-15 on page 4-23.

Figure 4-23 shows the ID_MMFR2 bit assignments.

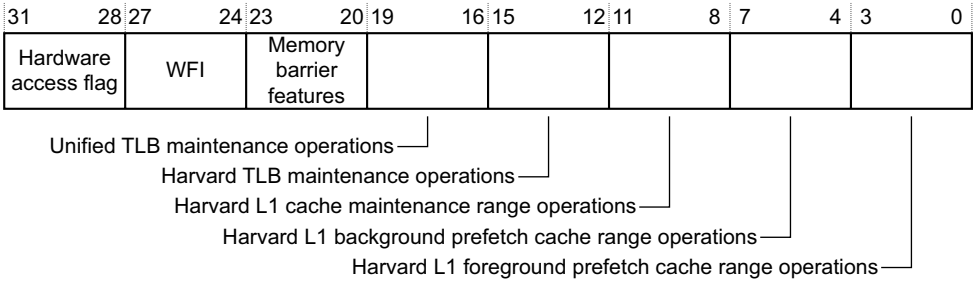


Figure 4-23 Memory Model Feature Register 2 bit assignments

Table 4-37 shows the ID_MMFR2 bit assignments.

Table 4-37 ID_MMFR2 bit assignments

Bits	Name	Description
[31:28]	Hardware Access Flag	Not supported.
[27:24]	Wait for interrupt stalling	Wait for interrupt supported.
[23:20]	Memory barrier features	• Data Synchronization Barrier • Instruction Synchronization Barrier • DataMemoryBarrier.
[19:16]	TLB maintenance operation, unified	• Invalidate all entries • Invalidate TLB entry by VA • Invalidate TLB entries by ASID match. • Invalidate TLB entries by VA and all ASID.
[15:12]	TLB maintenance operation, Harvard	Not supported.

Table 4-37 ID_MMFR2 bit assignments (continued)

Bits	Name	Description
[11:8]	L1 cache maintenance range operation, Harvard	Not supported.
[7:4]	L1 background prefetch cache range operations, Harvard	Not supported.
[3:0]	L1 foreground prefetch cache range operation, Harvard	Not supported.

To access the ID_MMFR2 use:

```
MRC p15, 0,<Rd>, c0, c1, 6
```

4.3.13 Memory Model Features Register 3

The ID_MMFR3 characteristics are:

Purpose Provides information about the memory model, memory management, cache support, and TLB operations of the processor.

Usage constraints The ID_MMFR3 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 4-15 on page 4-23.

Figure 4-24 shows the ID_MMFR3 bit assignments.

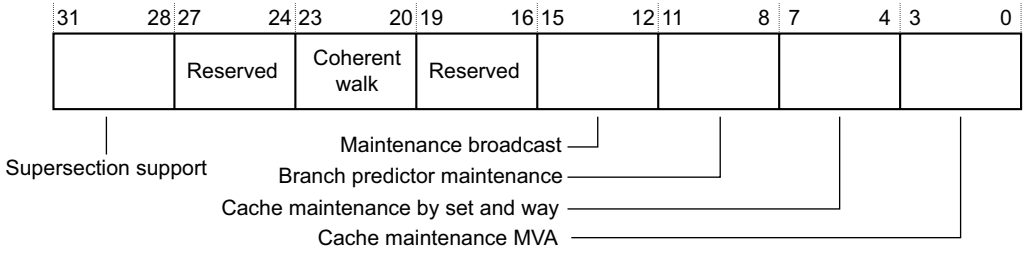


Figure 4-24 Memory Model Feature Register 3ID_MMFR3 bit assignments

Table 4-38 shows the ID_MMFR3 bit assignments.

Table 4-38 ID_MMFR3 bit assignments

Bits	Name	Description
[31:28]	Supersection support	Supported.
[27:24]	Reserved	RAZ.
[23:20]	Coherent walk	Supported.
[19:16]	Reserved	RAZ.
[15:12]	Maintenance broadcast	Supported.
[11:8]	Branch predictor maintenance	Supported.
[7:4]	Cache maintenance by Set/Way	Supported.
[3:0]	Cache maintenance VA	Supported.

To access the ID_MMFR3 use:

```
MRC p15, 0,<Rd>, c0, c1, 7
```

4.3.14 Instruction Set Attributes Register 0

The ID_ISAR0 characteristics are:

Purpose Provides information about the instruction set that the processor supports beyond the basic set.

Usage constraints The ID_ISAR0 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 4-15 on page 4-23.

Figure 4-25 on page 4-49 shows the ID_SAR0 bit assignments.

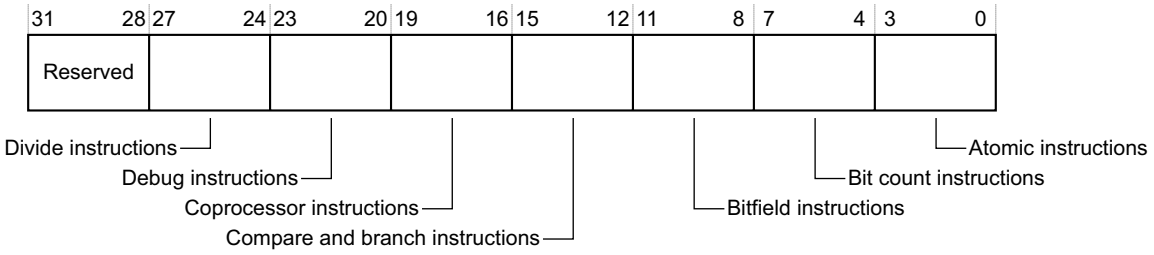


Figure 4-25 ID_SAR0 bit assignments

Table 4-39 shows the ID_ISAR0 bit assignments.

Table 4-39 ID_ISAR0 bit assignments

Bits	Name	Description
[31:28]	Reserved	RAZ.
[27:24]	Divide instructions	Not supported.
[23:20]	Debug instructions	`BKPT`.
[19:16]	Coprocessor instructions	Not supported - except for separately attributed architectures including CP15, CP14, Advanced SIMD instructions, and VFP.
[15:12]	CmpBranch instructions	Supported for CBZ and CBNZ.
[11:8]	Bitfield_instructions	Supported for BFC, BFI, SBFX, and UBFX.
[7:4]	BitCount_instructions	CLZ.
[3:0]	Swap instructions	SWP, SWPB.

To access the ID_ISAR0 use:

```
MRC p15, 0,<Rd>, c0, c2, 0
```

4.3.15 Instruction Set Attributes Register 1

The ID_ISAR1 characteristics are:

Purpose Provides information about the instruction set that the processor supports beyond the basic set.

Usage constraints The ID_ISAR1 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 4-15 on page 4-23.

Figure 4-26 shows the ID_ISAR1 bit assignments.

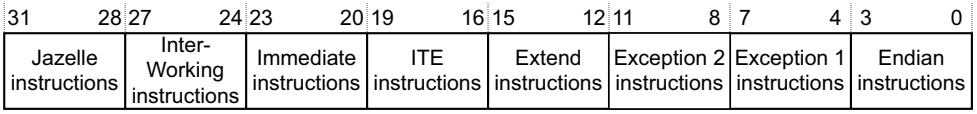


Figure 4-26 ID_ISARI bit assignments

Table 4-40 shows the ID_ISAR1 bit assignments.

Table 4-40 ID_ISARI bit assignments

Bits	Name	Description
[31:28]	Jazelle instructions	BXJ and J bit in PSRs
[27:24]	Interwork instructions	`BX`, `BLX`, T bit in PSRs PC loads and `DP` instructions have `BX`-like behavior
[23:20]	Immediate instructions	Special immediate-generating instructions supported
[19:16]	IfThen instructions	Supported
[15:12]	Extend instructions	All supported
[11:8]	Exception2 instructions	`SRS`, `RFE`, `CPS`
[7:4]	Exception1 instructions	(`LDM(2)`, `LDM(3)`, `STM(2)`
[3:0]	Endian instructions	`SETEND`

To access the ID_ISAR1, use:

```
MRC p15, 0,<Rd>, c0, c2, 1
```

4.3.16 Instruction Set Attributes Register 2

The ID_ISAR2 characteristics are:

Purpose Provides information about the instruction set that the processor supports beyond the basic set.

Usage constraints The ID_ISAR2 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 4-15 on page 4-23.

Figure 4-27 shows the ID_ISAR2 bit assignments.

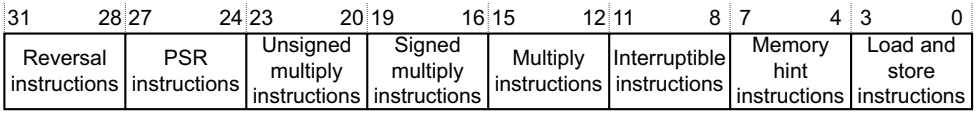


Figure 4-27 ID_ISAR2 bit assignments

Table 4-41 shows the ID_SAR2 bit assignments.

Table 4-41 ID_ISAR2 bit assignments

Bits	Name	Description
[31:28]	Reversal instructions	(REV, REV16, REVSH, and RBIT)
[27:24]	PSR instructions	(MRS, MSR and exception return data-processing instructions
[23:20]	Multiply instructions (advanced, unsigned)	(UMULL, UMLAL, and UMAAL)
[19:16]	Multiply instructions (advanced, signed)	(SMULL, SMAL, SMLABB, SMLABT, SMLALBB, SMLALBT, SMLALTB, SMLALTT, SMLATB, SMLATT, SMLAWB, SMLAWT, SMULBB, SMULBT, SMULTB, SMULTT, SMULWB, SMULWT, and Q flag in PSRs)
[15:12]	Multiply instructions	(MUL, MLA, and MLS)
[11:8]	Multi-access interruptible instructions	(non-interruptible)
[7:4]	Memory hint instructions	(PLD, PLI, PLDW)
[3:0]	Load and store instructions	(adds LDRD or STRD)

To access the ID_ISAR2 use:

```
MRC p15, 0,<Rd>, c0, c2, 2
```

4.3.17 Instruction Set Attributes Register 3

The ID_ISAR3 characteristics are:

Purpose Provides information about the instruction set that the processor supports beyond the basic set.

Usage constraints The ID_ISAR3 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 4-15 on page 4-23.

Figure 4-28 shows the ID_ISAR3 bit assignments.

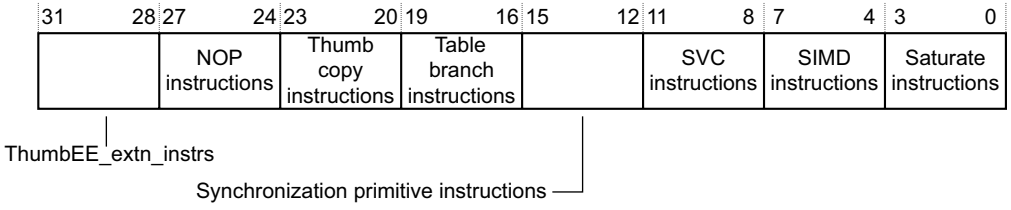


Figure 4-28 ID_ISAR3 bit assignments

Table 4-42 shows the ID_ISAR3 bit assignments.

Table 4-42 ID_ISAR3 bit assignments

Bits	Name	Description
[31:28]	Thumb-2 Executable Environment Extension instructions	Thumb-2 Executable Environment Extension instructions supported
[27:24]	True NOP instructions	NOP32
[23:20]	ThumbCopy instructions	Thumb `MOV(3)` low reg -> low reg and `CPY` alias
[19:16]	TableBranch instructions	`TBB`, `TBH`
[15:12]	SyncPrim instructions	`LDREX`, `STREX`, `LDRBEX`, `STRBEX`, `LDRHEX`, `STRHEX`, `LDRDEX`, `STRDEX`, `CLREX`

Table 4-42 ID_ISAR3 bit assignments (continued)

Bits	Name	Description
[11:8]	SVC instructions	Supported
[7:4]	SIMD instructions	All supported
[3:0]	Saturate instructions	QADD, QDADD, QDSUB, QSUB, and Q flag in PSRs

To access the ID_ISAR3 use:

```
MRC p15, 0,<Rd>, c0, c2, 3
```

4.3.18 Instruction Set Attributes Register 4

The ID_ISAR4 characteristics are:

Purpose Provides information about the instruction set that the processor supports beyond the basic set.

Usage constraints The ID_ISAR4 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 4-15 on page 4-23.

Figure 4-29 shows the ID_ISAR4 bit assignments.

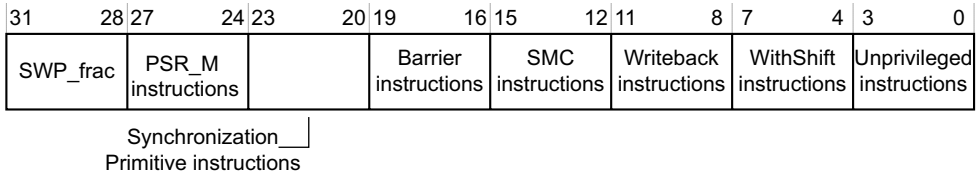


Figure 4-29 ID_ISAR4 bit assignments

Table 4-43 shows the ID_ISAR4 bit assignments.

Table 4-43 ID_ISAR4 bit assignments

Bits	Name	Description
[31:28]	SWP_frac	This field is only valid if Swap_instrs field in ID_ISAR0 == 0b0000. See *Instruction Set Attributes Register 1* on page 4-49.
[27:24]	PSR_M_instrs	Indicates the supported M profile instructions to modify the PSRs: None supported
[23:20]	Synchronization Primitive instructions	Indicates the supported Synchronization Primitive instructions: None supported
[19:16]	Barrier instructions	Indicates support for Barrier instructions: The processor supports `DMB`, `DSB`, and `ISB`.
[15:12]	SMC instructions	Indicates support for *Secure Monitor Call* (SMC) instructions: The processor supports `SMC`.
[11:8]	Write-back instructions	Indicates support for write-back instructions: The processor supports all defined write-back addressing modes.
[7:4]	With-shift instructions	Indicates support for instructions with shifts. The processor supports: • shifts of loads and stores over the range LSL 0-3 • constant shift options • register-controlled shift options.
[3:0]	Unprivileged instructions	Indicates support for Unprivileged instructions: The processor supports `LDR{SB|B|SH|H}T`.

To access the ID_ISAR4 use:

```
MRC p15, 0,<Rd>, c0, c2, 4
```

4.3.19 Instruction Set Attribute Registers 5-7

Instruction Set Attributes Registers 5-7 are reserved, and they read as `0x00000000`.

4.3.20 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 4-16 on page 4-25.

Figure 4-30 shows the CCSIDR bit assignments.

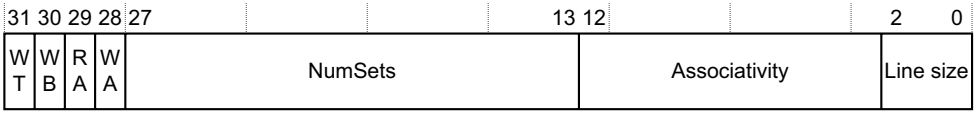


Figure 4-30 CCSIDR bit assignments

Table 4-44 shows how the CSSIDR bit assignments.

Table 4-44 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 4-44 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 4-58.

4.3.21 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 4-16 on page 4-25.

Figure 4-31 on page 4-57 shows the CLIDR bit assignments.

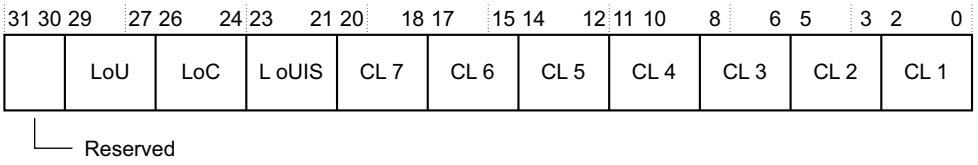


Figure 4-31 CLIDR bit assignments

Table 4-45 shows the CLIDR bit assignments.

Table 4-45 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
```

4.3.22 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 4-16 on page 4-25.

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.

4.3.23 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 4-15 on page 4-23.

Figure 4-32 shows the CSSELR bit assignments.

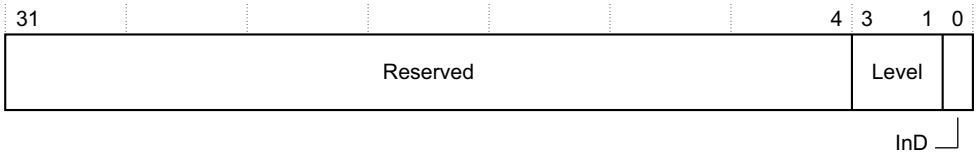


Figure 4-32 CSSELR bit assignments

Table 4-46 shows the CSSELR bit assignments.

Table 4-46 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
```

4.3.24 System Control Register

The SCTLR characteristics are:

Purpose Provides control and configuration of:

- memory alignment and endianness,
- memory protection and fault behavior
- MMU and cache enables
- interrupts and behavior of interrupt latency
- location for exception vectors
- program flow prediction.

Usage constraints The SCTLR is:

- only accessible in privileged modes.
- partially banked. Table 4-47 on page 4-60 shows banked and secure modify only bits.

Configurations Available in all configurations.

Attributes See the register summary in Table 4-16 on page 4-25.

Figure 4-33 on page 4-60 shows the SCTLR bit assignments.

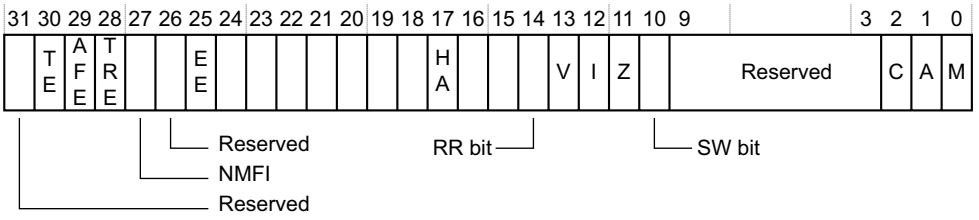


Figure 4-33 SCTLR bit assignments

Table 4-47 shows the SCTLR bit assignments.

Table 4-47 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 fast interrupt enable. The reset value is determined by **CFGNMFI**. The pin cannot be configured by software. This bit is read-only. 0 = FIQ exceptions can be masked in the CPSR. 1 = FIQ exceptions cannot be masked.
[26]	-	-	RAZ/SBZP

Table 4-47 SCTLR bit assignments (continued)

Bits	Name	Access	Description
[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`.When the Security Extensions are implemented 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 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 4-47 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, see *Security extensions write access disable* on page 3-12.

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
```

4.3.25 Auxiliary Control Register

The ACTLR characteristics are:

Purpose Controls

- parity checking, if implemented
- 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 features:
  - Allocation in one way
  - Full 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 4-16 on page 4-25.

Figure 4-34 shows the ACTLR bit assignments.

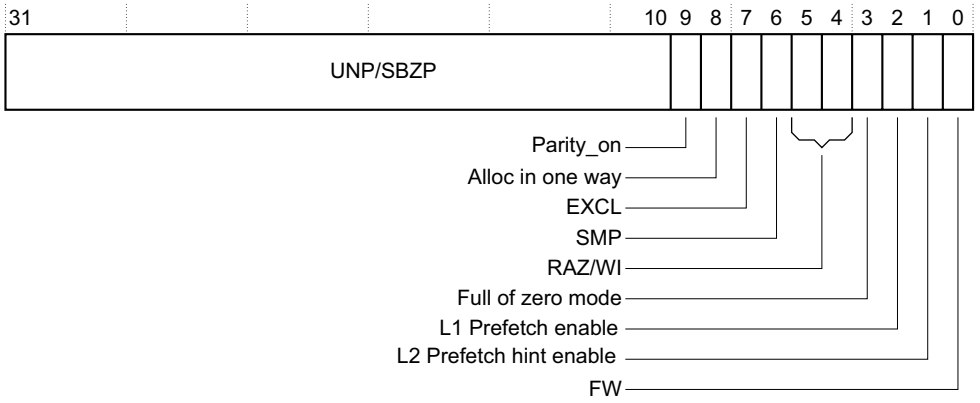


Figure 4-34 ACTLR bit assignments

Table 4-48 shows the ACTLR bit assignments.

Table 4-48 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. 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 4-48 ACTLR bit assignments (continued)

Bits	Name	Description
[5:4]	-	RAZ/WI
[3]	Full of zero mode	Enable full of zero mode. The reset value is zero.[a]
[2]	L1 prefetch enable	Dside prefetch. 0 = disabled. This is the reset value. 1 = enabled.
[1]	L2 prefetch enable	Prefetch hint enable. The reset value is zero.[a]
[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 ACTLR
MCR p15, 0,<Rd>, c1, c0, 1 ; Write ACTLR
```

Attempts to write to this register in secure privileged mode when **CP15SDISABLE** is HIGH result in an Undefined instruction exception, see *Security extensions write access disable* on page 3-12.

4.3.26 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 4-16 on page 4-25.

Figure 4-35 shows the CPACR bit assignments.

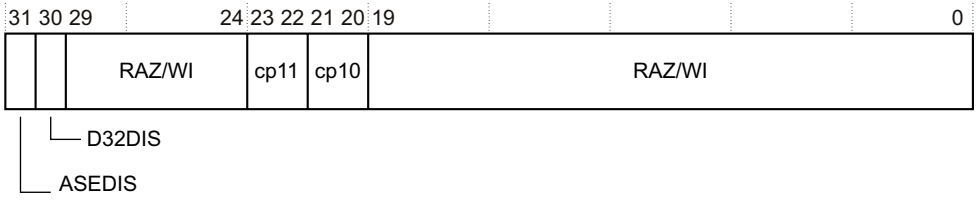


Figure 4-35 CPACR bit assignments

Table 4-49 shows the CPACR bit assignments.

Table 4-49 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 4-49 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 4-71.

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 4-71.

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.

4.3.27 Secure Configuration Register

The SCR characteristics are:

Purpose Controls:

- the current state of the processor as Secure or Non-secure
- the state the core executes exceptions in
- the ability to modify the A and I bits in the CPSR in the Non-secure state.

Usage constraints The SCR is:

- only accessible in privileged modes.
- only accessible in Secure state.

Configurations Available in all configurations.

Attributes See the register summary in Table 4-16 on page 4-25.

Figure 4-36 shows the SCR bit assignments.

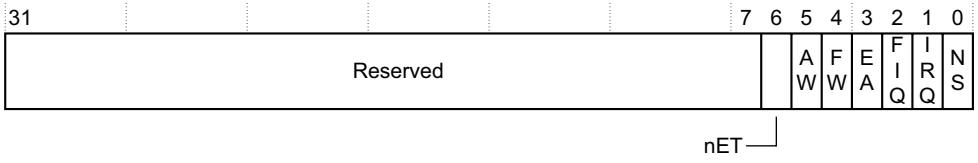


Figure 4-36 SCR bit assignments

Table 4-50 shows the SCR bit assignments.

Table 4-50 SCR bit assignments

Bits	Name	Description
[31:7]	-	Reserved
[6]	nET	Not early termination. RAZ/WI.
[5]	AW	Determines if the A bit in the CPSR can be modified when in the Non-secure state: 0 = disable modification of the A bit in the CPSR in the Non-secure state. This is the reset value. 1 = enable modification of the A bit in the CPSR in the Non-secure state.
[4]	FW	Determines if the F bit in the CPSR can be modified when in the Non-secure state: 0 = disable modification of the F bit in the CPSR in the Non-secure state. This is the reset value. 1 = enable modification of the F bit in the CPSR in the Non-secure state.
[3]	EA	Determines external abort behavior for Secure and Non-secure states: 0 = branch to abort mode on an external abort exception. This is the reset value. 1 = branch to Monitor mode on an external abort exception. When this bit is set to 1, and an external abort causes entry to Monitor mode, fault information is written to the Secure versions of the Fault Status and Fault Address registers.
[2]	FIQ	Determines FIQ behavior for Secure and Non-secure states: 0 = branch to FIQ mode on an FIQ exception. This is the reset value. 1 = branch to Monitor mode on an FIQ exception.
[1]	IRQ	Determines IRQ behavior for Secure and Non-secure states: 0 = branch to IRQ mode on an IRQ exception. This is the reset value. 1 = branch to Monitor mode on an IRQ exception.
[0]	NS bit	Defines the operation of the processor: 0 = secure. This is the reset value. 1 = Non-secure.

Note

When the processor runs in Monitor mode the state is considered Secure regardless of the state of the NS bit.

The values of the bits in the SCR have security implications. Table 4-51 shows the results for combinations of the FW and FIQ bits.

Table 4-51 Operation of the FW and FIQ bits

FW	FIQ	Description
1	0	FIQs handled locally.
0	1	FIQs can be configured to give deterministic secure interrupts.
1	1	Non-secure state able to make denial of service attack. Avoid use of this function.
0	0	Avoid because the core might enter an infinite loop for Non-secure FIQ.

Table 4-52 shows the results for combinations of the AW and EA bits.

Table 4-52 Operation of the AW and EA bits

AW	EA	Description
1	0	Aborts handled locally.
0	1	All external aborts trapped to Monitor mode.
1	1	All external asynchronous Data Aborts trapped to Monitor mode but the Non-secure state can hide secure aborts from the Monitor. Avoid use of this function.
0	0	Avoid because the core can unexpectedly enter an abort mode in the Non-secure state.

An attempt to access the SCR from any state other than secure privileged results in an undefined instruction exception.

To access the SCR, use:

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

4.3.28 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 4-16 on page 4-25.

Figure 4-37 shows the SDER bit assignments.

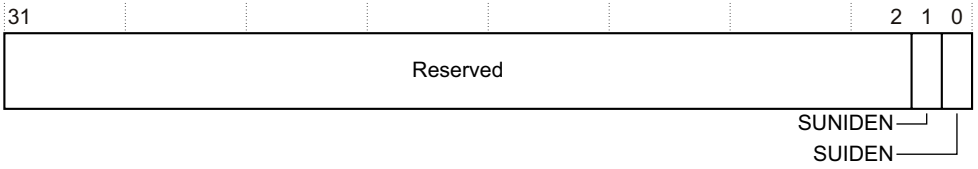


Figure 4-37 SDER bit assignments

Table 4-53 shows the SDER bit assignments.

Table 4-53 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
```

4.3.29 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 4-16 on page 4-25.

Figure 4-38 shows the NSACR bit assignments.

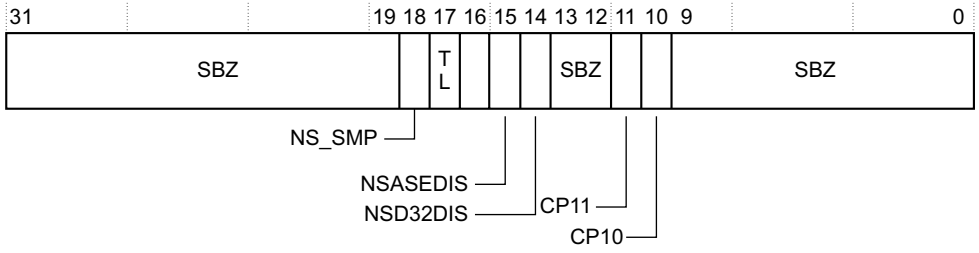


Figure 4-38 NSACR bit assignments

Table 4-54 shows the NSACR bit assignments.

Table 4-54 NSACR bit assignments

Bits	Name	Description
[31:19]	-	UNP or 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]	-	RAZ/WI

Table 4-54 NSACR bit assignments (continued)

Bits	Name	Description
[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]	-	UNP or SBZP.
[11]	CP11	Determines permission to access coprocessor 11 in the Non-secure state: 0 = Secure access only. This is the reset value. 1 = Secure or Non-secure access.
[10]	CP10	Determines permission to access coprocessor 10 in the Non-secure state: 0 = Secure access only. This is the reset value. 1 = Secure or Non-secure access.
[9:0]	-	UNP or SBZ

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.

4.3.30 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 4-25 on page 4-31.

Figure 4-39 shows the VCR bit assignments.

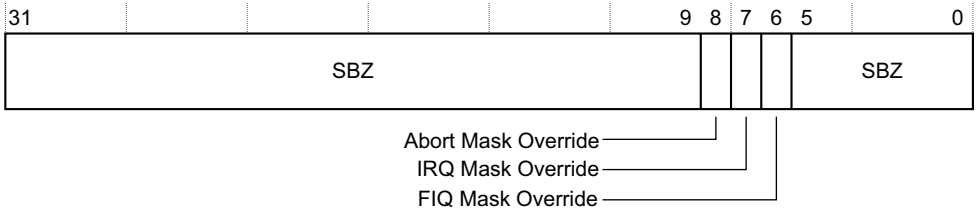


Figure 4-39 VCR bit assignments

Table 4-55 shows the VCR bit assignments.

Table 4-55 VCR bit assignments

Bits	Name	Description
[31:9]	-	Reserved
[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. See *Secure Configuration Register* on page 4-68.

Table 4-55 VCR bit assignments (continued)

Bits	Name	Description
[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. See *Secure Configuration Register* on page 4-68.
[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. See *Secure Configuration Register* on page 4-68.
[5:0]	-	Reserved

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
```

4.3.31 Translation Table Base Register 0

The TTBR0 characteristics are:

Purpose Holds the physical address of the first level translation table.

Usage constraints The TTBR0 is:

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

Attempts to write to this register in secure privileged mode when **CP15SDISABLE** is HIGH result in an Undefined instruction exception. See *Security extensions write access disable* on page 3-12.

Configurations Available in all configurations.

Attributes See the register summary in Table 4-17 on page 4-25.

Figure 4-40 shows the TTBR0 bit assignments. For an explanation of N in the figure, see *Translation Table Base Control Register* on page 4-79.

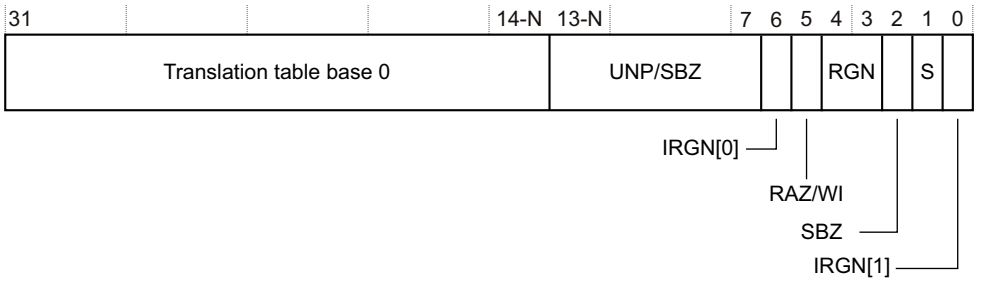


Figure 4-40 TTBR0 bit assignments

Table 4-56 shows the TTBR0 bit assignments. For an explanation of N in the table see *Translation Table Base Control Register* on page 4-79.

Table 4-56 TTBR0 bit assignments

Bits	Name	Description
[31:14-N]	Translation table base 0	Pointer to the level one translation table.
[13-N:7]	-	UNP or SBZ.
[6]	IRGN[0]	Used with bit 0, IRGN[1] to describe inner cacheability.
[5]	-	RAZ/WI.
[4:3]	RGN	Outer Cacheable attributes for translation table walking: b00 = Outer Non-cacheable b01 = Outer Cacheable Write-Back cached, Write-Allocate b10 = Outer Cacheable Write-Through, no allocate on write b11 = Outer Cacheable Write-Back, no allocate on write.

Table 4-56 TTBR0 bit assignments (continued)

Bits	Name	Description
[2]	-	SBZ. This bit is not implemented on this processor.
[1]	S	Translation table walk: 0 = to non-shared memory. 1 = to shared memory.
[0]	IRGN[1]	Indicates inner cacheability for the translation table walk: IRGN[1], IRGN[0] 00 = Noncacheable 01 = Write-Back Write-Allocate 10 = Write-Through, no allocate on write 11 = Write-Back no allocate on write. Page table walks do look-ups in the data cache only in write-back. Write-through is treated as noncacheable.

A write to the TTBR0 updates the address of the first level translation table from the value in bits [31:7] of the written value, to account for the maximum value of 7 for N. The number of bits of this address that the processor uses, and therefore the required alignment of the first level translation table, depends on the value of N, see *Translation Table Base Control Register* on page 4-79.

A read from the TTBR0 returns the complete address of the first level translation table in bits [31:7] of the read value, regardless of the value of N.

To access TTBR0, use:

```
MRC p15, 0,<Rd>, c2, c0, 0; Read Translation Table Base Register 0TTBR0
MCR p15, 0,<Rd>, c2, c0, 0; Write Translation Table Base Register 0TTBR0
```

4.3.32 Translation Table Base Register 1

The TTBR1 characteristics are:

Purpose Holds the physical address of the first-level translation table.

Usage constraints TTBR1 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 4-17 on page 4-25.

Figure 4-41 shows the TTBR1 bit assignments.

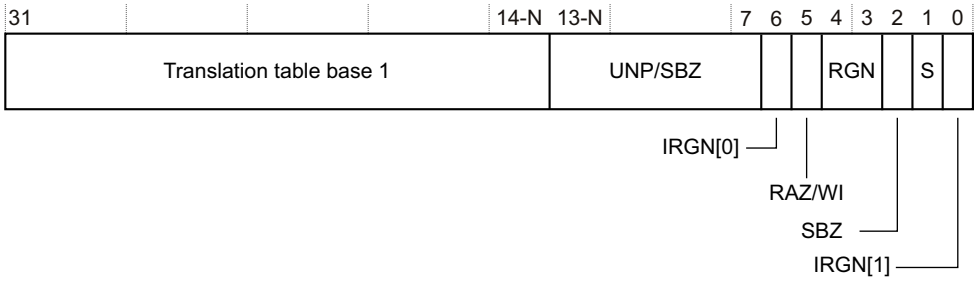


Figure 4-41 TTBR1 bit assignments

Table 4-57 shows the TTBR1 bit assignments.

Table 4-57 TTBR1 bit assignments

Bits	Name	Description
[31:14]	Translation table base 1	Pointer to the level one translation table.
[13:7]	-	UNP or SBZ
[6]	IRGN[0]	Used with IRGN[1] to describe inner cacheability.
[5]	-	RAZ/WI
[4:3]	RGN	Outer cacheable attributes for translation table walking: b00 = Outer Non-cacheable b01 = Outer Cacheable Write-Back cached, Write-Allocate b10 = Outer cacheable Write-Through, no allocate on write b11 = Outer cacheable write-back, no allocate on write.

Table 4-57 TTBR1 bit assignments (continued)

Bits	Name	Description
[2]	-	SBZ
[1]	S	Translation table walk: 1 = to Shared memory 0 = to non-shared memory.
[0]	IRGN[1]	Indicates inner cacheability for the translation table walk: IRGN[1], IRGN[0] 00 = Non-cacheable 01 = Write-Back Write-Allocate 10 = Write-Through, no allocate on write 11 = Write-Back no allocate on write. Page table walks do look-ups in the data cache only in write-back. Write-through is treated as noncacheable.

To access TTBR1, use:

```
MRC p15, 0,<Rd>, c2, c0, 1; Read TTBR1
MCR p15, 0,<Rd>, c2, c0, 1; Write TTBR1
```

Writing to CP15 c2 updates the pointer to the first level translation table from the value in bits [31:14] of the written value. Bits [13:7] Should Be Zero. The address specified by TTBR1 must reside on a 16KB page boundary.

4.3.33 Translation Table Base Control Register

The TTBCR characteristics are:

Purpose Determines which of the Translation Table Base Registers, TTBR0 or TTBR1, defines the base address for the translation table walk that is required when a VA is not found in the TLB.

Usage constraints The TTBCR is:

- only accessible in privileged modes.
- banked.

Attempts to write to this register in secure privileged mode when **CP15SDISABLE** is HIGH result in an Undefined instruction exception. See *Security extensions write access disable* on page 3-12.

This register has a defined reset value of 0. This reset value applies only to the Secure copy of the register, and software must program the Non-secure copy of the register with the required value

Configurations Available in all configurations.

Attributes See the register summary in Table 4-17 on page 4-25.

Figure 4-42 shows the TTBCR bit assignments.

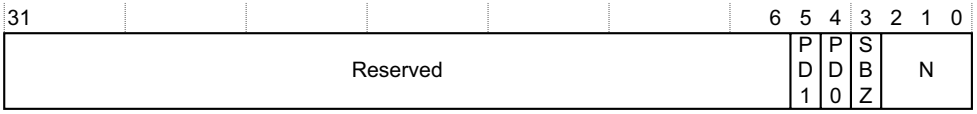


Figure 4-42 TTBCR bit assignments

Table 4-58 shows the TTBCR bit assignments.

Table 4-58 TTBCR bit assignments

Bits	Name	Description
[31:6]	-	UNP or SBZ.
[5]	PD1	Specifies occurrence of a translation table walk on a TLB miss when using TTBR1. When translation table walk is disabled, a section translation fault occurs instead on a TLB miss: 0 = The processor performs a translation table walk on a TLB miss, with secure or Non-secure privilege appropriate to the current Secure or Non-secure state. This is the reset value. The Non-secure version of this register must be programmed by software. 1 = The processor does not perform a translation table walk. If a TLB miss occurs with Translation Table Base Register 1 in use, the processor returns a section translation fault.

Table 4-58 TTBCR bit assignments (continued)

Bits	Name	Description
[4]	PD0	Specifies occurrence of a translation table walk on a TLB miss when using TTBR0. When translation table walk is disabled, a section translation fault occurs instead of a TLB miss. 0 = The processor performs a translation table walk on a TLB miss, with secure or Non-secure privilege appropriate to the current Secure or Non-secure state. This is the reset value. The Non-secure version of this register must be programmed by software. 1 = The processor does not perform a translation table walk. If a TLB miss occurs with Translation Table Base Register 0 in use, the processor returns a section translation fault.
[3]	-	UNP or SBZ.
[2:0]	N	Specifies the boundary size of TTBR0. b000 = 16KB. This is the reset value. b001 = 8KB b010 = 4KB b011 = 2KB b100 = 1KB b101 = 512 bytes b110 = 256 bytes b111 = 128 bytes.

To access the TTBCR, use:

```
MRC p15, 0,<Rd>, c2, c0, 2 ; Read TTBCR
MCR p15, 0,<Rd>, c2, c0, 2 ; Write TTBCR
```

A translation table base register is selected as follows:

- If N is set to 0, always use TTBR0. This is the default case at reset for the Secure version of this register. It is backwards compatible with ARMv5 and earlier processors.
- If N is set greater than 0, and bits [31:32-N] of the VA are all zeros, use TTBR0, otherwise use TTBR1. N must be in the range 0-7.

Note

The Cortex-A9 processor can perform a translation table walk from L1 cache. So, page tables placed in inner write-back memory do not require any clean operation after being modified.

4.3.34 Domain Access Control Register

The DACR characteristics are:

Purpose Holds the access permissions for a maximum of 16 domains.

Usage constraints The DACR is:

- only accessible in privileged modes.
- banked.

Attempts to write to this register in secure privileged mode when **CP15SDISABLE** is HIGH result in an Undefined instruction exception, see *Security extensions write access disable* on page 3-12.

Configurations Available in all configurations.

Attributes See the register summary in Table 4-18 on page 4-26.

Figure 4-43 DACR bit assignments.

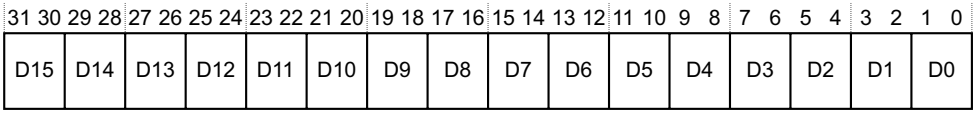


Figure 4-43 DACR bit assignments

Table 4-59 shows how the bit values correspond with the DACR functions.

Table 4-59 DACR bit assignments

Bits	Name	Description
-	D<n>[a]	The fields D15-D0 in the register define the access permissions for each one of the 16 domains. b00 = No access. Any access generates a domain fault. b01 = Client. Accesses are checked against the access permission bits in the TLB entry. b10 = Reserved. Any access generates a domain fault. b11 = Manager. Accesses are not checked against the access permission bits in the TLB entry, so a permission fault cannot be generated. Attempting to execute code in a page that has the TLB *eXecute Never* (XN) attribute set does not generate an abort.

a. n is the Domain number in the range between 0 and 15.

To access the DACR, use:

```
MRC p15, 0,<Rd>, c3, c0, 0 ; Read DACR
```

```
MCR p15, 0,<Rd>, c3, c0, 0 ; Write DACR
```

4.3.35 Data Fault Status Register

The DFSR characteristics are:

Purpose
- Holds the source of the last data fault.
- Indicates the domain and type of access being performed when an abort occurred

Usage constraints The DFSR is:
- only accessible in privileged modes.
- banked.

Configurations Available in all configurations.

Attributes See the register summary in Table 4-19 on page 4-26.

Figure 4-44 shows the DFSR bit assignments.

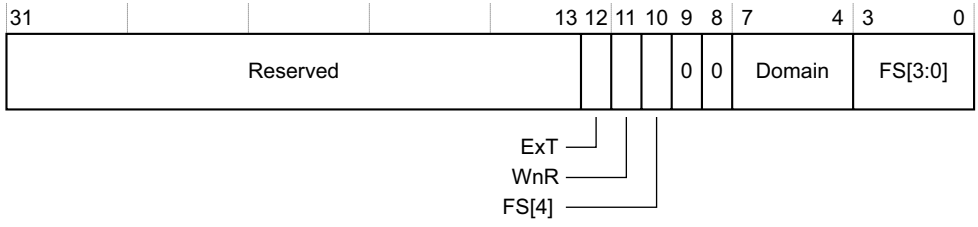


Figure 4-44 DFSR bit assignments

Table 4-60 shows the DFSR bit assignments.

Table 4-60 DFSR bit assignments

Bits	Name	Description
[31:13]		UNP or SBZ.
[12]	ExT	External Abort Qualifier. Indicates whether an AXI Decode or Slave error caused an abort. This bit is only valid for External Aborts. For all other aborts this bit *Should Be Zero*. 0= external abort marked as DECERR[a] 1= external abort marked as SLVERR
[11]	WnR	Not read and write. Indicates what type of access caused the abort: 0 = read 1 = write. In case of aborted CP15 operations, this bit is set to 1.
[10]	FS[4]	Part of the Status field. See bit [12] and bit[3:0] in this table.

Table 4-60 DFSR bit assignments (continued)

Bits	Name	Description
[9:8]	-	Always read as 0.
[7:4]	Domain	Specifies which of the 16 domains, D15-D0, was being accessed when a data fault occurred.
[3:0]	Status	Indicates the type of exception generated. To determine the data fault, bits [12] and [10] must be used in conjunction with bits[3:0]. The following encodings are in priority order, highest first: 1. b000001 alignment fault 2. b000100 instruction cache maintenance fault 3. b*x*01100 1st level translation, synchronous external abort 4. b*x*01110 2nd level translation, synchronous external abort 5. b000101 translation fault, section 6. b000111 translation fault, page 7. b000011 access flag fault, section 8. b000110 access flag fault, page 9. b001001 domain fault, section 10. b001011 domain fault, page 11. b001101 permission fault, section 12. b001111 permission fault, page 13. b*x*01000 synchronous external abort, nontranslation 14. b*x*10110 asynchronous external abort 15. b000010 debug event. Any unused encoding not listed is reserved. Where *x* represents bit [12] in the encoding, bit [12] can be either: 0 = AXI Decode error caused the abort. This is the reset value. 1 = AXI Slave error caused the abort.

a. SLVERR and DECERR are the two possible types of abort reported in an AXI bus.

To access the DFSR, use:

```
MRC p15, 0,<Rd>, c5, c0, 0; Read DFSR
MCR p15, 0,<Rd>, c5, c0, 0; Write DFSR
```

Reading CP15 c5 with Opcode_2 set to 0 returns the value of the DFSR.

Writing CP15 c5 with Opcode_2 set to 0 sets the DFSR to the value of the data written. This is useful for a debugger to restore the value of the DFSR. The register must be written using a read modify write sequence.

4.3.36 Instruction Fault Status Register

The IFSR characteristics are:

Purpose
- Holds the source of the last instruction fault.
- Indicates the domain and type of access being performed when an abort occurred

Usage constraints The IFSR is:
- only accessible in privileged modes.
- banked.

Configurations Available in all configurations.

Attributes See the register summary in Table 4-19 on page 4-26.

Figure 4-45 shows the IFSR bit assignments.

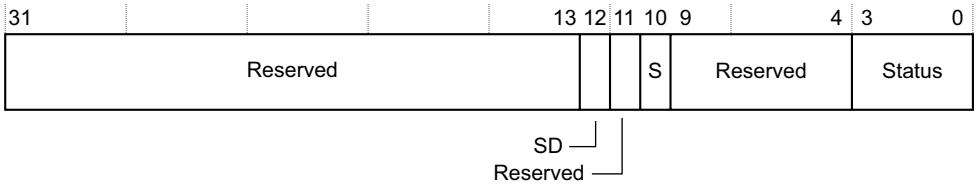


Figure 4-45 IFSR bit assignments

Table 4-61 shows the IFSR bit assignments.

Table 4-61 IFSR bit assignments

Bits	Name	Description
[31:13]	-	UNP or SBZ.
[12]	SD	External abort qualifier 0 = External abort marked as DECERR 1 = External abort marked as SLVERR.
[11]	-	Always reads as 0.

Table 4-61 IFSR bit assignments (continued)

Bits	Name	Description
[10]	S	Part of the status field. Must be used in conjunction with bit[12] and bits[3:0]
[9:4]	-	Always reads as 0.
[3:0]	Status	Type of fault generated.Indicates the type of exception generated. To determine the data fault, bits [12] and [10] must be used in conjunction with bits[3:0]. The following encodings are in priority order, with 1 being highest: 1. b*x*01100 L1 translation, synchronous external abort 2. b*x*01110 L2 translation, synchronous external abort 3. b000101 translation fault, section 4. b000111 translation fault, page 5. b000011 access flag fault, section 6. b000110 access flag fault, page 7. b001001 domain fault, section 8. b001011 domain fault, page 9. b001101 permission fault, section 10. b001111 permission fault, page 11. b*x*01000 synchronous external abort 12. b000010 debug event. Any unused encoding not listed is reserved. Where *x* represents bit [12] in the encoding, bit [12] can be either: 0 = AXI Decode error caused the abort. This is the reset value. 1 = AXI Slave error caused the abort.

You can access the IFSR by reading or writing CP15 c5 with the Opcode_2 field set to 1:

```
MRC p15, 0,<Rd>, c5, c0, 1; Read Instruction Fault Status Register
MCR p15, 0,<Rd>, c5, c0, 1; Write Instruction Fault Status Register
```

Note

When the SCR.EA bit is set the processor writes to the Secure Data Fault Status Register on a Monitor entry caused by an External Abort. See *Secure Configuration Register* on page 4-68.

Reading CP15 c5 with the Opcode_2 field set to 1 returns the value of the IFSR.

Writing CP15 c5 with the Opcode_2 field set to 1 sets the IFSR to the value of the data written. This is useful for a debugger to restore the value of the IFSR. The register must be written using a read, modify, write sequence. Bits [31:4] Should Be Zero.

4.3.37 Auxiliary Data Fault Status Register

The ADFSR characteristics are:

Purpose Can provide implementation specific information.

Usage constraints The ADFSR 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 4-19 on page 4-26.

Note

The ADFSR is unused in this implementation.

To access the ADFSR, use:

```
MRC p15, 0, <Rd>, c5, c1, 0; Read Data Auxiliary Fault Status Register
MCR p15, 0, <Rd>, c5, c1, 0; Write Data Auxiliary Fault Status Register
```

4.3.38 Auxiliary Instruction Fault Status Register

The AIFSR characteristics are:

Purpose Can provide implementation specific information.

Usage constraints The AIFSR 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 4-19 on page 4-26.

Note

The AIFSR is unused in this implementation.

To access the AIFSR, use

```
MRC p15, 0, <Rd>, c5, c1, 1; Read Instruction Auxiliary Fault Status Register
MCR p15, 0, <Rd>, c5, c1, 1; Write Instruction Auxiliary Fault Status Register
```

The Auxiliary Fault Status Registers are provided for compatibility with all ARMv7-A designs. The processor always reads this as RAZ. All writes are ignored.

There is no physical register for the Auxiliary Data Fault Status Register or Auxiliary Instruction Fault Status Register because the register is always RAZ.

4.3.39 Data Fault Address Register

The DFAR characteristics are:

Purpose Holds the faulting address when a synchronous fault occurs.

Usage constraints The DFAR 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 4-20 on page 4-26.

To access the DFAR, use:

```
MRC p15, 0, <Rd>, c6, c0, 0 ; Read Data Fault Address Register

MCR p15, 0, <Rd>, c6, c0, 0 ; Write Data Fault Address Register
```

A write to this register sets the DFAR to the value of the data written. This is useful for a debugger to restore the value of the DFAR.

The Cortex-A9 processor does not update the DFAR on debug exception entry if in debug monitor mode. In halting debug mode the Cortex-A9 processor does not update the DFAR on debug exception entry. In both cases the WFAR is updated.See Chapter 8 *Debug*.

4.3.40 Instruction Fault Address Register

The IFAR characteristics are:

Purpose holds the address of the instruction that caused a prefetch abort.

Usage constraints The IFAR 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 4-20 on page 4-26.

To access the IFAR, use:

```
MRC p15, 0, <Rd>, c6, c0, 2 ; Read Instruction Fault Address Register

MCR p15, 0, <Rd>, c6, c0, 2 ; Write Instruction Fault Address Register
```

A write to this register sets the IFAR to the value of the data written. This is useful for a debugger to restore the value of the IFAR. For more information, see *Breakpoints and watchpoints* on page 8-6.

4.3.41 NOP Register

The use of this registers is optional and deprecated. Use the `NOP` instruction instead.

4.3.42 Physical Address Register

The PAR characteristics are:

Purpose Holds

- the PA after a successful translation
- the source of the abort for an unsuccessful translation.

Usage constraints The PAR 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 4-21 on page 4-27.

The PA Register format depends on the value of bit F This bit signals whether or not there is an error during the VA to PA translation. See *VA format* on page 4-10 for information on Virtual Addresses.

The PA Register is accessed by reading to CP15 c7 with <CRm> field set to c4 and Opcode_2 field set to 0:

```
MRC p15,0,<Rd>,c7,c4,0; Read PA register
```

If the translation aborted, the FSR bits give the encoding of the source of the abort as shown in Figure 4-46 on page 4-91. See *Data Fault Status Register* on page 4-83 and *Instruction Fault Status Register* on page 4-86 for information on the Fault Status Register bits and the SD bit. Figure 4-46 on page 4-91 shows the bit assignment for PAR aborted translations.

Figure 4-46 PAR aborted translation bit assignments

If the translation completed successfully, the PAR bit assignment is as Figure 4-47 shows.

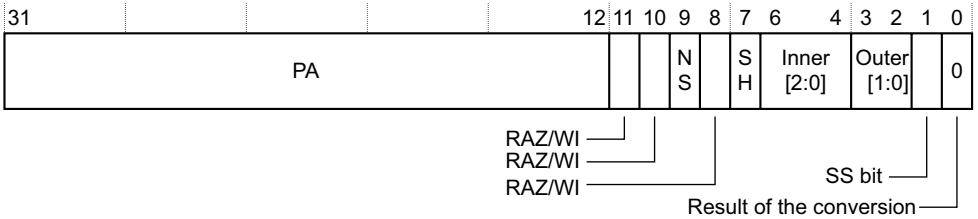


Figure 4-47 PAR successful translation bit assignments

Table 4-62 shows the PAR bit assignments.

Table 4-62 PAR bit assignments

Bits	Name	Description
[31:12]	PA	Physical address.
[11]	-	RAZ/WI
[10]	-	RAZ/WI
[9]	NS	Non-secure. The NS bit from the translation table entry.
[8]	-	RAZ/WI
[7]	SH	Shareable attribute. Indicates whether the physical memory is shareable: 0 = Memory is non-shareable. 1 = Memory is shareable.

Table 4-62 PAR bit assignments (continued)

Bits	Name	Description
[6:4]	Inner	Signals region inner attributes: b000 =Inner noncacheable. b001 = Strongly-ordered. b011 = Device. b101 = Inner Write-Back Write-Allocate. b110 = Inner Write-Through. No Write-Allocate (treated as inner Non-cacheable). b111 =Inner Write-Back. No Write-Allocate (treated as Write-Allocate).
[3:2]	Outer	Signals region outer attributes for normal memory type (type = b10): b00 = Outer Non-cacheable. b01 = Outer Write-Back Write-Allocate. b10 = Outer Write-Through. No Write-Allocate. b11 = Outer Write-Back. No Write-Allocate.
[1]	SS bit	Supersection bit: 0 = The translation is not a supersection. 1 = The translation is a supersection.
[0]	F	0 = Successful translation.

4.3.43 Instruction Synchronization Barrier

The use of ISB is optional and deprecated. Use the instruction `ISB` instead.

4.3.44 Data Synchronization Barrier

The use of DSB is optional and deprecated. Use the instruction `DSB` instead.

4.3.45 Data Memory Barrier

The use of DMB is optional and deprecated. Use the instruction `DMB` instead.

4.3.46 Performance Monitor Control Register

The PMCR Register characteristics are:

Purpose Controls operation of the

- Performance Monitor Count Registers

- Cycle Counter Register.

Usage constraints The PMCR is:

- common to Secure and Non-secure states.
- accessible as determined by *User Enable Register* on page 4-110.

Configurations Available in all configurations.

Attributes See the register summary in Table 4-22 on page 4-29.

Figure 4-48 shows the PMCR bit assignments.

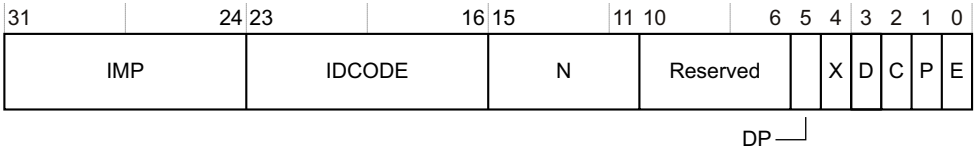


Figure 4-48 PMCR bit assignments

Table 4-63 shows the PMCR bit assignments.

Table 4-63 PMCR bit assignments

Bits	Name	Description
[31:24]	IMP	Specifies the implementor code: 0x41 = ARM. This field is read-only and write ignored.
[23:16]	IdCode	Specifies the identification code: 0x09 his field is read-only and write ignored.
[15:11]	N	Specifies the number of counters implemented: b00110 = 6 counters implemented. This field is read-only and write ignored.
[10:6]	-	RAZ/SBZP
[5]	DP	Disables cycle counter, CCNT, when prohibited: 0 = count is enabled in prohibited regions. This is the reset value. 1 = count is disabled in prohibited regions.

Table 4-63 PMCR bit assignments (continued)

Bits	Name	Description
[4]	X	Enables export of the events from the event bus to an external monitoring block, such as a PTM: 0 = export disabled. This is the reset value. 1 = export enabled.
[3]	D	Cycle count divider: 0 = count every clock cycle when enabled. This is the reset value. 1 = count every 64th clock cycle when enabled.
[2]	C	Cycle counter reset, write only bit, RAZ: 0 = no action 1 = reset cycle counter, CCNT, to zero.
[1]	P	Performance counter reset, write only bit, RAZ: 0 = no action 1 = reset all performance counters to zero, not including CCNT.
[0]	E	Enable bit: 0 = disable all counters, including CCNT. This is the reset value. 1 = enable all counters including CCNT.

The PMCR is always accessible in privileged modes.

To access the PMNC Register, use:

```
MRC p15, 0,<Rd>, c9, c12, 0 ; Read PMNC Register
MCR p15, 0,<Rd>, c9, c12, 0 ; Write PMNC Register
```

4.3.47 Count Enable Set Register

The PMCNTENSET characteristics are:

Purpose The PMCNTENSET:

- enables PMCCNT
- enables PMNx
- indicates which counters are enabled.

Usage constraints The PMCNTENSET is:

- common to Secure and Non-secure states.
- accessible as determined by *User Enable Register* on page 4-110.

Configurations Available in all configurations.

Attributes See the register summary in Table 4-22 on page 4-29.

Figure 4-49 shows the PMCNTENSET bit assignments.

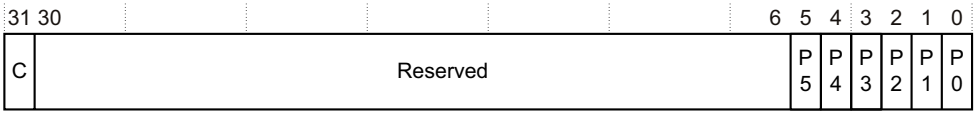


Figure 4-49 PMCNTENSET bit assignments

Table 4-64 shows the PMCNTENSET bit assignments.

Table 4-64 PMCNTENSET bit assignments

Bits	Name	Description
[31]	C	Cycle counter enable set: 0 = disable 1 = enable.
[30:6]	-	UNP or SBZP
[5]	P5	Counter 5 enable
[4]	P4	Counter 4 enable
[3]	P3	Counter 3 enable
[2]	P2	Counter 2 enable
[1]	P1	Counter 1 enable
[0]	P0	Counter 0 enable

To access the PMCNTENSET Register, use:

```
MRC p15, 0,<Rd>, c9, c12, 1 ; Read PMCNTENSET Register
MCR p15, 0,<Rd>, c9, c12, 1 ; Write PMCNTENSET Register
```

When reading this register, any enable that reads as 0 indicates the counter is disabled. An enable that reads as 1 indicates the counter is enabled.

When writing this register, any enable written with a value of 0 is ignored, that is, not updated. An enable written with a value of 1 indicates the counter is enabled.

4.3.48 Count Enable Clear Register

The PMCNTENCLR characteristics are:

Purpose Disables:

- the Cycle Count Register.
- any implemented event counters.

Indicates counters that are enabled.

Usage constraints The PMCNTENCLR is:

- common to Secure and Non-secure states.
- accessible as determined by *User Enable Register* on page 4-110.

Configurations Available in all configurations.

Attributes See the register summary in Table 4-22 on page 4-29.

Figure 4-50 shows the PMCNTENCLR bit assignments.

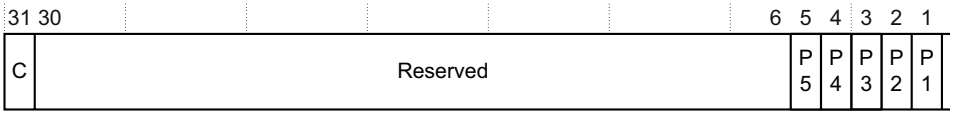


Figure 4-50 PMCNTENCLR bit assignments

Table 4-65 shows the PMCNTENCLR bit assignments.

Table 4-65 PMCNTENCLR bit assignments

Bits	Name	Description
[31]	C	Cycle counter enable clear: 0 = disable 1 = enable.
[30:6]	-	Reserved
[5]	P5	Counter 5 enable
[4]	P4	Counter 4 enable
[3]	P3	Counter 3 enable

Table 4-65 PMCNTENCLR bit assignments (continued)

Bits	Name	Description
[2]	P2	Counter 2 enable
[1]	P1	Counter 1 enable
[0]	P0	Counter 0 enable

To access the PMCNTENCLR, use:

```
MRC p15, 0,<Rd>, c9, c12, 2 ; Read PMCNTENCLR Register
MCR p15, 0,<Rd>, c9, c12, 2 ; Write PMCNTENCLR Register
```

When reading this register, any enable that reads as 0 indicates the counter is disabled. An enable that reads as 1 indicates the counter is enabled.

When writing this register, any enable written with a value of 0 is ignored, that is, not updated. An enable written with a value of 1 clears the counter enable.

You can use the enable, EN, bit [0] of the PMNC Register to disable all performance counters including CCNT. The PMCNTENCLR retains its value when the enable bit of the PMNC is clear, even though its settings are ignored.

4.3.49 Overflow Flag Status Register

The PMOVSR characteristics are:

Purpose Holds the state of the overflow flags for:

- the Cycle Count Register.
- each of the implemented event counters.

Usage constraints The PMOVSR is:

- common to Secure and Non-secure states.
- accessible as determined by *User Enable Register* on page 4-110.

Configurations Available in all configurations.

Attributes See the register summary in Table 4-22 on page 4-29.

Figure 4-51 on page 4-98 shows the PMOVSR bit assignments.

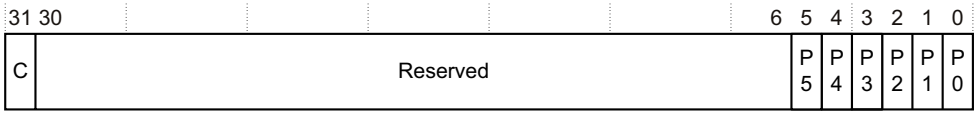


Figure 4-51 PMOVSR bit assignments

Table 4-66 shows the PMOVSR bit assignments.

Table 4-66 PMOVSR bit assignments

Bits	Name	Description
[31]	C	Cycle counter overflow flag: 0 = disable 1 = enable.
[30:6]	-	UNP or SBZP
[5]	P5	Counter 5 overflow flag
[4]	P4	Counter 4 overflow flag
[3]	P3	Counter 3 overflow flag
[2]	P2	Counter 2 overflow flag
[1]	P1	Counter 1 overflow flag
[0]	P0	Counter 0 overflow flag

To access the PMOVSR, use:

```
MRC p15, 0,<Rd>, c9, c12, 3 ; Read PMOVSR Register
MCR p15, 0,<Rd>, c9, c12, 3 ; Write PMOVSR Register
```

When reading this register, any overflow flag that reads as 0 indicates the counter has not overflowed. An overflow flag that reads as 1 indicates the counter has overflowed.

When writing this register, any overflow flag written with a value of 0 is ignored, that is, no change. An overflow flag written with a value of 1 clears the counter overflow flag.

4.3.50 Software Increment Register

The PMSWINC Register characteristics are:

Purpose Increments the count of a performance monitor count register.

Usage constraints The PMSWINC is:

- common to Secure and Non-secure states.
- accessible as determined by *User Enable Register* on page 4-110.

Configurations Available in all configurations.

Attributes See the register summary in Table 4-22 on page 4-29.

Figure 4-52 shows the PMSWINC bit assignments.

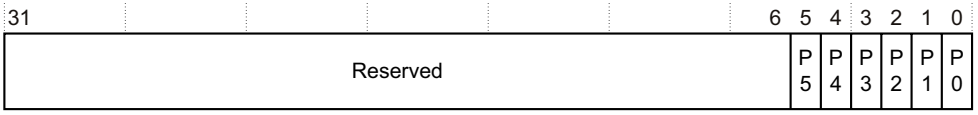


Figure 4-52 PMSWINC bit assignments

Table 4-67 shows the PMSWINC bit assignments.

Table 4-67 PMSWINC bit assignments

Bits	Name	Description
[31:4]	-	RAZ/SBZP
[5]	P5	Increment Counter 5
[4]	P4	Increment Counter 4
[3]	P3	Increment Counter 3
[2]	P2	Increment Counter 2
[1]	P1	Increment Counter 1
[0]	P0	Increment Counter 0

The PMSWINC only has effect when the counter event is set to `0x00`.

To access the PMSWINC Register, write CP15 with:

```
MCR p15, 0,<Rd>, c9, c12, 4 ; Write PMSWINC Register
```

When writing this register, a value of 1 increments the counter, and value of 0 does nothing.

4.3.51 Event Counter Selection Register

The PMSELR characteristics are:

Purpose Selects a Performance Monitor Count Register.

Usage constraints The PMSELR is:

- common to Secure and Non-secure states.
- accessible as determined by *User Enable Register* on page 4-110.

Configurations Available in all configurations.

Attributes See the register summary in Table 4-22 on page 4-29.

Figure 4-53 shows the PMSELR bit assignments.

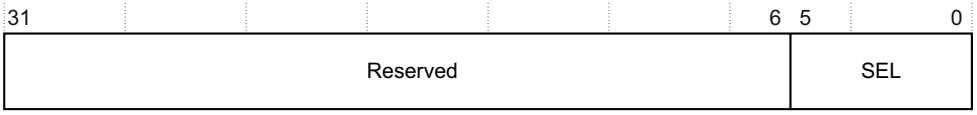


Figure 4-53 PMSELR bit assignments

Table 4-68 shows the PMSELR functions.

Table 4-68 PMSELR bit assignments

Bits	Name	Description
[31:6]	-	RAZ/SBZP
[5:0]	SEL	Counter select: 5'b00101 = selects counter 5 5'b00100 = selects counter 4 5'b00011 = selects counter 3 5'b00010 = selects counter 2 5'b00001 = selects counter 1 5'b00000 = selects counter 0.

Any values programmed in the PMSELR other than those specified in Table 4-68 are Unpredictable.

To access the PMSELR, use:

```
MRC p15, 0,<Rd>, c9, c12, 5; Read PMSELR
MCR p15, 0,<Rd>, c9, c12, 5; Write PMSELR
```

4.3.52 Cycle Count Register

The PMCCNTR characteristics are:

Purpose Counts instances of an event that a particular Performance Monitor Control Register can count.

Usage constraints The PMCCNTR is:

- common to Secure and Non-secure states.
- accessible as determined by *User Enable Register* on page 4-110.

Configurations Available in all configurations.

Attributes See the register summary in Table 4-22 on page 4-29.

To access the PMCCNTR Register, use:

```
MRC p15, 0,<Rd>, c9, c13, 0 ; Read PMCCNTR Register
MCR p15, 0,<Rd>, c9, c13, 0 ; Write PMCCNTR Register
```

The PMCCNTR Register must be disabled before software can write to it. Any attempt by software to write to this register when enabled is Unpredictable.

4.3.53 Event Selection Register

The PMXEVTYPER characteristics are:

Purpose Selects the events that you want a Performance Monitor Count Register to count.

Usage constraints The PMXEVTYPER is:

- common to Secure and Non-secure states
- accessible as determined by *User Enable Register* on page 4-110.

Configurations Available in all configurations.

Attributes See the register summary in Table 4-22 on page 4-29.

Figure 4-54 on page 4-102 shows the PMXEVTYPER bit assignments.

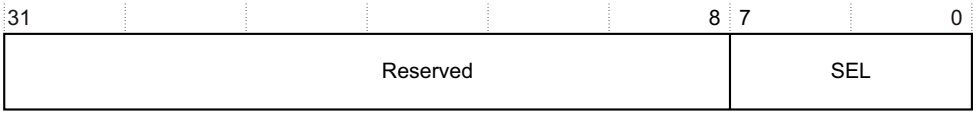


Figure 4-54 PMXEVTYPER bit assignments

Table 4-69 shows the PMXEVTYPER bit assignments.

Table 4-69 PMXEVTYPER bit assignments

Bits	Name	Description
[31:8]	-	RAZ/SBZP
[7:0]	SEL	Specifies the event selected as described in the *ARM Architecture Reference Manual*. Table 4-71 on page 4-104 and Table 4-72 on page 4-105, describes additional Cortex-A9 specific events.

To access the PMXEVTYPER Register, use:

```
MRC p15, 0,<Rd>, c9, c13, 1 ; Read PMXEVTYPER Register
MCR p15, 0,<Rd>, c9, c13, 1 ; Write PMXEVTYPER Register
```

Table 4-70 shows the predefined events in summary form. The *ARM Architecture Reference Manual* shows the range of values for predefined events that you can monitor using PMXEVTYPER.

Table 4-70 Predefined events summary

Event	Description
0x00	Software increment. The register is incremented only on writes to the Software Increment Register. See *Software Increment Register* on page 4-98.
0x01	Instruction fetch that causes a refill at (at least) the lowest level(s) of instruction or unified cache. Includes the speculative linefills in the count.
0x02	Instruction fetch that causes a TLB refill at (at least) the lowest level of TLB. Includes the speculative requests in the count
0x03	Data read or write operation that causes a refill at (at least) the lowest level(s) of data or unified cache. Counts the number of allocations performed in the Data Cache due to a read or a write.
0x04	Data read or write operation that causes a cache access at (at least) the lowest level(s) of data or unified cache. This includes speculative reads

Table 4-70 Predefined events summary (continued)

Event	Description
0x05	Data read or write operation that causes a TLB refill at (at least) the lowest level of TLB. This does not include micro TLB misses due to PLD, PLI, CP15 Cache operation by MVA and CP15 VA to PA operations.
0x06	Data read architecturally executed. Counts the number of data read instructions accepted by the Load Store Unit. This includes counting the speculative and aborted LDR/LDM, as well as the reads due to the SWP instructions.
0x07	Data write architecturally executed. Counts the number of data write instructions accepted by the Load Store Unit. This includes counting the speculative and aborted STR/STM, as well as the writes due to the SWP instructions.
0x08	Not used.
0x09	Exception taken. Counts the number of exceptions architecturally taken.
0x0A	Exception return architecturally executed. The following instructions are reported on this event: • `LDM pc [.]^` • `RFE` • `DP S pc`
0x0B	Change to ContextID retired. Counts the number of instructions architecturally executed writing into the ContextID Register.
0x0C	Software change of PC, except by an exception, architecturally executed. Count the number of PC changes architecturally executed, excluding the PC changes due to taken exceptions.
0x0D	Immediate branch architecturally executed (taken or not taken). This includes the branches which are flushed due to a previous load/store which aborts late.
0x0E	Not used.
0x0F	Unaligned access architecturally executed. Counts the number of aborted unaligned accessed architecturally executed, and the number of not-aborted unaligned accesses, including the speculative ones.
0x10	Branch mispredicted/not predicted. Counts the number of mispredicted or not-predicted branches executed. This includes the branches which are flushed due to a previous load/store which aborts late.

Table 4-70 Predefined events summary (continued)

Event	Description
0x11	Counts clock cycles when the Cortex-A9 processor is not in WFE/WFI. This event is not exported on the PMUEVENT bus.
0x12	Branches or other change in program flow that could have been predicted by the branch prediction resources of the processor. This includes the branches which are flushed due to a previous load/store which aborts late.
0x13-0x3F	Reserved.

If this unit generates an interrupt, the processor asserts the pin **PMUIRQ**. You can route this pin to an external interrupt controller for prioritization and masking. This is the only mechanism that signals this interrupt to the core.

The absolute counts recorded might vary because of pipeline effects. This has negligible effect except in cases where the counters are enabled for a very short time.

Cortex-A9 specific events

Table 4-71 and Table 4-72 on page 4-105 show events for Cortex-A9 processors. In the value column Precise means the event is counted precisely. Events related to stalls and speculative instructions appear as Approximate entries in this column.

Table 4-71 shows the Jazelle events.

Table 4-71 Jazelle events

Event	Description	Value
0x40	Java bytecode execute. Counts the number of Java bytecodes being decoded, including speculative ones.	Approximate
0x41	Software Java bytecode executed. Counts the number of software java bytecodes being decoded, including speculative ones	Approximate
0x42	Jazelle backward branches executed. Counts the number of Jazelle taken branches being executed. This includes the branches which are flushed due to a previous load/store which aborts late.	Approximate

Table 4-72 shows the Cortex-A9 specific events.

Table 4-72 Cortex-A9 specific events

Event	Description	Value
0x50	Coherent linefill miss[a] 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[a] 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 due to the instruction side not being able to provide any and the instruction cache is currently performing at least one linefill.	Approximate
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 due to the instruction side not being able to provide the instructions, or to the data side not being able to provide the necessary data, due to 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 due to 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

Table 4-72 Cortex-A9 specific events (continued)

Event	Description	Value
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 renamed • b01 one instruction renamed • b10 two instructions renamed. See Table A-17 on page A-16 for a description of how these values map to the PMUEVENT bus bits.	Approximate
0x6E	Predictable function returns Counts the number of procedure returns whose condition codes do not fail, excluding all returns from exception. This count includes procedure returns which are flushed due to 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

Table 4-72 Cortex-A9 specific events (continued)

Event	Description	Value
0x72	Load/Store Instructions Counts the number of instructions being executed in the Load/Store unit. The counted instructions are still speculative.	Approximate
0x73	Floating-point instructions Counts the number of Floating-point instructions going through the Register Rename stage. Instructions are still speculative in this stage. Two floating-point instructions can be renamed in the same cycle so the event is two bits long: 0b00 no floating-point instruction renamed 0b01 one floating-point instruction renamed 0b10 two floating-point instructions renamed. See Table A-17 on page A-16 for a description of how these values map to the PMUEVENT bus bits.	Approximate
0x74	NEON instructions Counts the number of Neon instructions going through the Register Rename stage. Instructions are still speculative in this stage. Two NEON instructions can be renamed in the same cycle so the event is two bits long: 0b00 no NEON instruction renamed 0b01 one NEON instruction renamed 0b10 two NEON instructions renamed. See Table A-17 on page A-16 for a description of how these values map to the PMUEVENT bus bits.	Approximate
0x80	Processor stalls due to PLDs Counts the number of cycles where the processor is stalled because PLD slots are all full	Approximate
0x81	Processor stalled due to 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 due to instruction side main TLB miss Counts the number of stall cycles due to main TLB misses on requests issued by the instruction side	Approximate
0x83	Processor stalled due to data side main TLB miss Counts the number of stall cycles due to main TLB misses on requests issued by the data side.	Approximate

Table 4-72 Cortex-A9 specific events (continued)

Event	Description	Value
0x84	Processor stalled due to instruction micro TLB miss Counts the number of stall cycles due to 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 due to data micro TLB miss Counts the number of stall cycles due to 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 due to DMB Counts the number of stall cycles due to 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
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

a. For use with Cortex-A9 multiprocessor variants.

DEFLAGS[6:0]

If your design includes a Data Engine you can use **DEFLAGS[6:0]** to export the values of the FPSCR error trapping bits. See *Performance monitoring signals* on page A-16.

4.3.54 Performance Monitor Count Registers

The PMCNT characteristics are:

Purpose The PMCNTs count instances of an event selected by the PMXEVTYPER Register. The bits of each PMCNT contain an event count.

Usage constraints The PMCNTs are:

- common to Secure and Non-secure states
- accessible as determined by *User Enable Register* on page 4-110.

Configurations There are six PMCNTs.

Attributes See the register summary in Table 4-22 on page 4-29.

To access the PMCNT Registers, use:

```
MRC p15, 0,<Rd>, c9, c13, 2 ; Read PMCNT0-PMCNT5 Registers
MCR p15, 0,<Rd>, c9, c13, 2 ; Write PMCNT0-PMCNT5 Registers
```

Table 4-73 shows what signal settings are required and the Secure or Non-secure state and mode that you can enable the counters.

Table 4-73 Signal settings for the Performance Monitor Count Registers

DBGEN	SPIDEN						
NIDEN	SPNIDEN	SUNIDEN	Secure state	User mode	PMNC[5]	Performance counters enabled	CCNT enabled
0	-	-	-	-	1'b0	No	Yes
0	-	-	-	-	1'b1	No	No
1	-	-	No	-	-	Yes	Yes
1	-	-	Yes	-	-	Yes	Yes
1	0	-	Yes	No	1'b0	No	Yes
1	0	-	Yes	No	1'b1	No	No
1	0	0	Yes	Yes	1'b0	No	Yes
1	0	0	Yes	Yes	1'b1	No	No
1	0	1	Yes	Yes	X	Yes	Yes

4.3.55 User Enable Register

The PMUSERENR characteristics are:

Purpose Enables user mode to have access to the Performance Monitor Registers.

Usage constraints The PMUSERENR is:

- writable only in privileged mode and readable in any processor mode.
- common to Secure and Non-secure states.

Note

The PMUSERENR does not provide access to the registers that control interrupt generation.

Configurations Available in all configurations.

Attributes See the register summary in Table 4-22 on page 4-29.

Figure 4-55 shows the PMUSERENR bit assignments.

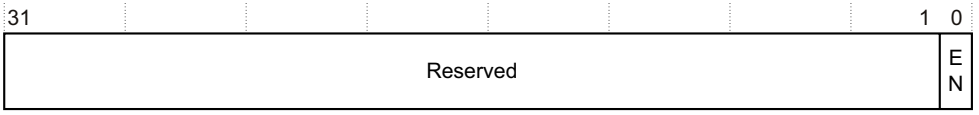


Figure 4-55 PMUSERENR bit assignments

Table 4-74 shows the PMUSERENR bit assignments.

Table 4-74 PMUSERENR bit assignments

Bits	Name	Description
[31:1]	-	RAZ/SBZP
[0]	EN	User mode enable. 0 is the reset value

To access the PMUSERENR, use:

```
MRC p15, 0,<Rd>, c9, c14, 0 ; Read PMUSERENR Register
MCR p15, 0,<Rd>, c9, c14, 0 ; Write PMUSERENR Register
```

4.3.56 Interrupt Enable Set Register

The PMINTENSET Register characteristics are:

Purpose Determines if the PMCNTs or CCNT generate an interrupt on overflow.

Usage constraints The PMINTENSET 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 4-22 on page 4-29.

Figure 4-56 shows the PMINTENSET bit assignments.

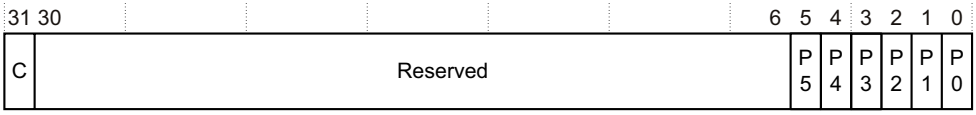


Figure 4-56 PMINTENSET bit assignments

Table 4-75 shows the PMINTENSET bit assignments.

Table 4-75 PMINTENSET bit assignments

Bits	Name	Description
[31]	C	CCNT overflow interrupt enable. When reading this register: 0 = interrupt disabled 1 = interrupt enabled. When writing to this register: 0 = no action 1 = interrupt enabled.
[30:6]	-	UNP or SBZP.
[5]	P5	PMCNT5 overflow interrupt enable.
[4]	P4	PMCNT4 overflow interrupt enable.
[3]	P3	PMCNT3 overflow interrupt enable.

Table 4-75 PMINTENSET bit assignments (continued)

Bits	Name	Description
[2]	P2	PMCNT2 overflow interrupt enable.
[1]	P1	PMCNT1 overflow interrupt enable.
[0]	P0	PMCNT0 overflow interrupt enable.

To access the PMINTENSET Register, use:

```
MRC p15, 0,<Rd>, c9, c14, 1 ; Read PMINTENSET Register
MCR p15, 0,<Rd>, c9, c14, 1 ; Write PMINTENSET Register
```

Reading this register returns the current setting. Writing to this register can enable interrupts. You can disable interrupts only by writing to the PMINTENCLR. See *Interrupt Enable Clear Register*.

4.3.57 Interrupt Enable Clear Register

The PMINTENCLR characteristics are:

Purpose Disables the generation of interrupt requests on overflows from:
- PMCNTs
- CCNT.

Usage constraints The PMINTENCLR is:
- only accessible in privileged modes.
- common to Secure and Non-secure state.

Configurations Available in all configurations.

Attributes See the register summary in Table 4-22 on page 4-29.

Figure 4-57 shows the PMINTENCLR bit assignments.

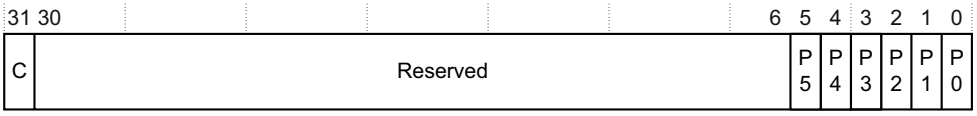


Figure 4-57 PMINTENCLR bit assignments

Table 4-76 shows the PMINTENCLR bit assignments.

Table 4-76 PMINTENCLR bit assignments

Bits	Name	Description
[31]	C	CCNT overflow interrupt enable bit. When reading this register: 0 = interrupt disabled 1 = interrupt enabled. When writing to this register: 0 = no action 1 = interrupt enabled.
[30:6]	-	UNP or SBZP.
[5]	P5	Interrupt on PMCNT5 overflow when enabled.
[4]	P4	Interrupt on PMCNT4 overflow when enabled.
[3]	P3	Interrupt on PMCNT3 overflow when enabled.
[2]	P2	Interrupt on PMCNT2 overflow when enabled.
[1]	P1	Interrupt on PMCNT1 overflow when enabled.
[0]	P0	Interrupt on PMCNT0 overflow when enabled.

To access the PMINTENCLR, use:

```
MRC p15, 0,<Rd>, c9, c14, 2 ; Read PMINTENCLR Register
MCR p15, 0,<Rd>, c9, c14, 2 ; Write PMINTENCLR Register
```

4.3.58 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 4-1 on page 4-4.

Figure 4-58 shows the TLB Lockdown Register bit assignments.

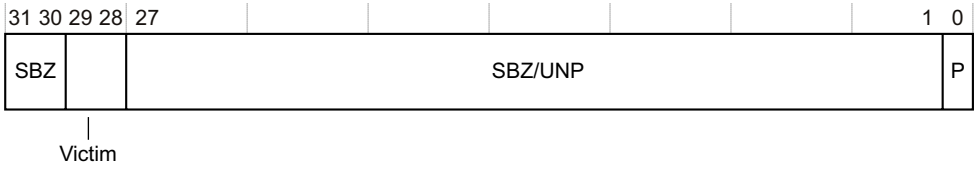


Figure 4-58 TLB Lockdown Register bit assignments

Table 4-77 shows the TLB Lockdown Register bit assignments

Table 4-77 TLB Lockdown Register bit assignments

Bits	Name	Description
[31:30]	-	SBZ
[29:28]	Victim	Lock down region
[27:1]	-	SBZ/UNP
[0]	P	Preserve bit. 0 is the reset value.

You can access the TLB Lockdown Register by reading or writing CP15 c10 with the Opcode_2 field set to 0:

```
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.

4.3.59 Primary Region Remap Register

The PRRR characteristics are:

Purpose Controls the top level mapping of the TEX[0], C, and B memory region attributes

Usage constraints The PRRR is:

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

Attempts to write to this register in secure privileged mode when **CP15SDISABLE** is HIGH result in an Undefined instruction exception, see *Security extensions write access disable* on page 3-12.

The PRRR is only used when TEX mapping is enabled by setting the SCTLR.TRE bit to 1.

Configurations Available in all configurations.

Attributes See the register summary in Table 4-22 on page 4-29.

Figure 4-59 shows the PRRR bit assignments.

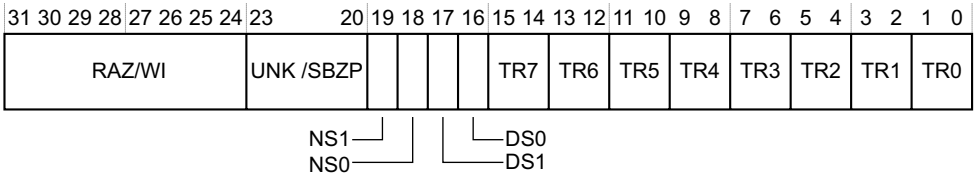


Figure 4-59 PRRR bit assignments

Table 4-78 shows the PRRR bit assignments.

Table 4-78 PRRR bit assignments

Bits	Name	Reset value	Description
[31:24]]	-	0	RAZ/WI
[23:20]	-	-	UNK/SBZP
[19]	NS1	1	Remaps shareable attribute when S = 1, for Normal regions
[18]	NS0	0	Remaps shareable attribute when S = 0, for Normal regions

Table 4-78 PRRR bit assignments (continued)

Bits	Name	Reset value	Description
[17]	DS1	0	Remaps shareable attribute when S = 1, for Device regions
[16]	DS0	1	Remaps shareable attribute when S = 0, for Device regions
[15:14]	TR7	10	Primary TEX mapping 1,1,1
[13:12]	TR6	00	RAZ/WI
[11:10]	TR5	10	Primary TEX mapping 1,0,1
[9:8]	TR4	10	Primary TEX mapping 1,0,0
[7:6]	TR3	10	Primary TEX mapping 0.1,1
[5:4]	TR2	10	Primary TEX mapping 0,1,0
[3:2]	TR1	01	Primary TEX mapping 0, 0,1
[1:0]	TR0	00	Primary TEX mapping0,0, 0

The reset values ensure that no remapping occurs at reset.

The Shareable bit can also be remapped. If the Shared bit as read from the TLB or translation tables is 0, then it is remapped to bit [15] of this register. If the Shared bit as read from the TLB or translation tables is 1, then it is remapped to bit [16] of this register. See *AXI USER bits* on page 7-4.

4.3.60 Normal Memory Remap Register

The NMRR characteristics are:

Purpose Provides additional mapping controls for memory regions that are mapped as Normal memory by their entry in the PRRR.

Usage constraints The NMRR is:

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

Attempts to write to this register in secure privileged mode when **CP15SDISABLE** is HIGH result in an Undefined instruction exception, see *Security extensions write access disable* on page 3-12.

The NMRR is only used when TEX mapping is enabled by setting the SCTLR.TRE bit to 1. This register gives the Outer Cacheable and Inner Cacheable property mapping for memory attributes, if the region is mapped as Normal Memory by the TRn entry in the PRRR.

Configurations Available in all configurations.

Attributes See the register summary in Table 4-1 on page 4-4.

Figure 4-60 shows the NMRR bit assignments.

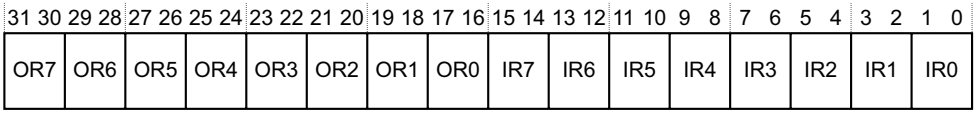


Figure 4-60 NMRR bit assignments

Table 4-79 shows the NMRR bit assignments.

Table 4-79 NMRR bit assignments

Bits	Name	Reset Value	Meaning
[31:30]	OR7	01	Remaps Outer cacheable property mapping for TR7
[29:28]	OR6	00	RAZ/WI
[27:26]	OR5	01	Remaps Outer cacheable property mapping for TR5
[25:24]	OR4	00	Remaps Outer cacheable property mapping for TR4
[23:22]	OR3	11	Remaps Outer cacheable property mapping for TR3
[21:20]	OR2	10	Remaps Outer cacheable property mapping for TR2
[19:18]	OR1	00	Remaps Outer cacheable property mapping for TR1
[17:16]	OR0	00	Remaps Outer cacheable property mapping for TR0
[15:14]	IR7	01	Remaps Inner cacheable property mapping for TR7
[13:12]	IR6	00	RAZ/WI
[11:10]	IR5	10	Remaps Inner cacheable property mapping for TR5
[9:8]	IR4	00	Remaps Inner cacheable property mapping for TR4
[7:6]	IR3	11	Remaps Inner cacheable property mapping for TR3

Table 4-79 NMRR bit assignments (continued)

Bits	Name	Reset Value	Meaning
[5:4]	IR2	10	Remaps Inner cacheable property mapping for TR2
[3:2]	IR1	00	Remaps Inner cacheable property mapping for TR1
[1:0]	IR0	00	Remaps Inner cacheable property mapping for TR0

The reset value for each field means that no remapping occurs.

The remap registers are expected to be static throughout operation. In particular, when the remap registers are changed, it is implementation-defined when the changes take effect. It is expected that an invalidation of the TLB and an Instruction Memory Barrier must be performed before any change of the Remap registers can be relied on.

The reset value for each field ensures that by default no remapping occurs.

Table 4-80 shows the condition of the memory regions, or types, when the MMU is disabled prior to remapping.

Table 4-80 Default memory regions when MMU is disabled

Condition	Region type
Data Cache enabled	Data, Strongly-ordered
Data Cache disabled	Data, Strongly-ordered
Instruction Cache enabled	Instruction, Write-Back, Write-Allocate
Instruction Cache disabled	Instruction, Strongly-ordered

This enables different mappings to be selected with the MMU disabled, that cannot be done using only the I, C, and M bits in CP15 c1. See *System Control Register* on page 4-59.

4.3.61 Vector Base Address Register

The VBAR characteristics are:

Purpose Holds the exception base address for exception vectors that are not handled in Monitor mode.

Usage constraints The VBAR is:

- only accessible in privileged modes.

- banked for Secure and Non-secure states.

Attempts to write to this register in secure privileged mode when **CP15SDISABLE** is HIGH result in an Undefined instruction exception, see *Security extensions write access disable* on page 3-12.

Configurations Available in all configurations.

Attributes See the register summary in Table 4-1 on page 4-4.

Figure 4-61 shows the bit arrangement of the VBAR.

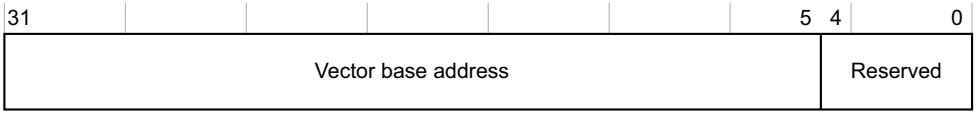


Figure 4-61 VBAR bit assignments

Table 4-81 shows how the bit values correspond with the VBAR functions.

Table 4-81 VBAR bit assignments

Bits	Name	Description
[31:5]	Vector base address	Holds the base address. Determines the location that the core branches to, on an exception. The reset value for the Secure version of this register is 0. The Non-secure Vector Base Address Register must be reset by the programmer before an exception occurs.
[4:0]	-	Reserved.

When an exception occurs in the Secure state, the core branches to address:

Secure Vector_Base_Address + Exception_Vector_Address.

When an exception occurs in the Non-secure state, the core branches to address:

Non-secure Vector_Base_Address + Exception_Vector_Address.

When high vectors are enabled, regardless of the value of the register the processor branches to:

`0xFFFF0000` + Exception_Vector_Address.

You can configure IRQ, FIQ, and external abort exceptions to branch to Monitor mode, see *Secure Configuration Register* on page 4-68. In this case, the processor uses the Monitor Vector Base Address, see *Monitor Vector Base Address Register*, to calculate the branch address. The Reset exception always branches to `0x00000000`, regardless of the value of the Vector Base Address except when the processor uses high vectors.

To access the VBAR, use:

```
MRC p15, 0, <Rd>, c12, c0, 0 ; Read Secure or Non-secure Vector Base Address
                             ; Register
MCR p15, 0, <Rd>, c12, c0, 0 ; Write Secure or Non-secure Vector Base Address
                             ; Register
```

4.3.62 Monitor Vector Base Address Register

The MVBAR characteristics are:

Purpose Holds the base address for the Monitor mode exception vector.

Usage constraints The MVBAR is:

- a read and write register in the Secure state only
- accessible in secure privileged modes only.

Note

Attempts to write to this register in secure privileged mode when **CP15SDISABLE** is HIGH result in an Undefined instruction exception. See *Security extensions write access disable* on page 3-12.

Configurations Available in all configurations.

Attributes See the register summary in Table 4-22 on page 4-29.

Figure 4-62 shows the MVBAR bit assignments.

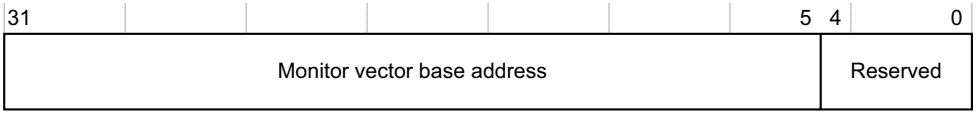


Figure 4-62 MVBAR bit assignments

Table 4-82 shows how the MVBAR bit assignments.

Table 4-82 MVBAR bit assignments

Bits	Name	Description
[31:5]	Monitor vector base address	Holds the base address. Determines the location that the core branches to, on an exception.
[4:0]	-	UNP or SBZ.

When an exception branches to the Monitor mode, the core branches to address:

Monitor_Base_Address + Exception_Vector_Address.

The Software Monitor Exception caused by an `SMC` instruction branches to Monitor mode. You can configure IRQ, FIQ, and External abort exceptions to branch to Monitor mode, see *Secure Configuration Register* on page 4-68. These are the only exceptions that can branch to Monitor mode and that use the Monitor Vector Base Address Register to calculate the branch address.

Note

The secure boot code must program the register with an appropriate value for the Monitor.

To access the MVBAR, use:

```
MRC p15, 0, <Rd>, c12, c0, 1 ; Read Monitor Vector Base Address Register
MCR p15, 0, <Rd>, c12, c0, 1 ; Write Monitor Vector Base Address Register
```

4.3.63 Interrupt Status Register

The Interrupt Status Register characteristics are:

Purpose Indicates the status of interrupt bits.

Usage constraints The Interrupt Status Register is:

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

Configurations Available in all configurations.

Attributes See the register summary in Table 4-22 on page 4-29.

Figure 4-63 on page 4-122 shows the Interrupt Status Register bit assignments.

Figure 4-63 Interrupt Status Register bit assignments

Table 4-83 shows how the bit values correspond with the Interrupt Status Register functions.

Table 4-83 Interrupt Status Register bit assignments

Bits	Name	Description
[31:9]	-	-
[8]	A	External abort pending flag 0 no pending external abort 1 in Secure state, an external abort is pending 1 in Non-secure state, an external abort or a virtual external abort is pending.
[7]	I	Interrupt pending flag 0 no pending IRQ 1in Secure state, an IRQ interrupt is pending 1 in Non-secure state an IRQ interrupt or a virtual IRQ interrupt is pending.
[6]	F	Fast interrupt pending flag 0 no pending FIQ 1 in Secure state, an FIQ fast interrupt is pending 1 in Non-secure state, an FIQ fast interrupt, or a virtual FIQ is pending.
[5:0]	-	Reserved

To access the Interrupt Status Register, use:

```
MRC p15, 0, <Rd>, c12, c1, 0 ; Read Interrupt Status Register
```

4.3.64 Virtualization Interrupt Register

The Virtualization Interrupt Register characteristics are:

Purpose Indicates that there is a virtual interrupt pending.

Usage constraints The Virtualization Interrupt Register 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 on page 4-30.

The virtual interrupt is delivered as soon as the processor is in NS state. Figure 4-64 shows the Virtualization Interrupt Register bit assignments.

Figure 4-64 Virtualization Interrupt Register bit assignments

Table 4-84 shows the Virtualization Interrupt Register bit assignments.

Table 4-84 Virtualization Interrupt Register bit assignments

Bits	Name	Description
[31:9]	-	SBZ.
[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]		SBZ.

To access the Virtualization Interrupt Register, use:

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

4.3.65 FCSE PID Register

The register is present but has no effect.The use of the FCSE PID Register is optional and deprecated. See *Context ID Register* on page 4-124.

4.3.66 Context ID Register

The CONTEXTIDR characteristics are:

Purpose Provides information on the current ASID and process ID.

Usage constraints The CONTEXTIDR 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 4-26 on page 4-31.

Figure 4-65 shows the CONTEXTIDR bit assignments.

Figure 4-65 CONTEXTIDR bit assignments

Table 4-85 shows the CONTEXTIDR bit assignments.

Table 4-85 CONTEXTIDR bit assignments

Bits	Name	Description
[31:8]	PROCID	Process Identifier
[7:0]	ASID	Address Space Identifier

To access the Context ID Register, use:

```
MRC p15, 0,<Rd>, c13, c0, 1; Read Context ID Register
MCR p15, 0,<Rd>, c13, c0, 1; Write Context ID Register
```

The bottom eight bits of the Context ID Register are used for the current ASID that is running. The top bits extend the ASID. To ensure that all accesses are related to the correct context ID, you must ensure that software executes a Data Synchronization Barrier operation before changing this register.

The value of this register can also be used to enable process-dependent breakpoints and watchpoints. After changing this register, an ISB sequence must be executed before any instructions are executed that are from an ASID-dependent memory region. Code that updates the ASID must be executed from a global memory region. See *Breakpoints and watchpoints* on page 8-6.

Note

Changing the Context ID Register value means that the virtual to physical address mapping is changed, so the *Branch Target Address Cache* (BTAC) must be flushed.

4.3.67 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 4-26 on page 4-31.

Figure 4-66 shows the Power Control Register bit assignments.

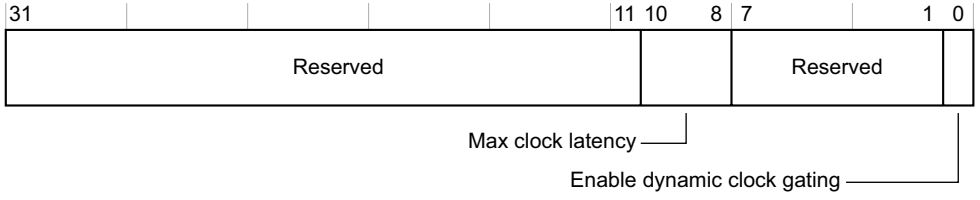


Figure 4-66 Power Control Register bit assignments

Table 4-86 shows the Power Control Register bit assignments.

Table 4-86 Power Control Register bit assignments

Bits	Name	Description
[31:11]	-	Reserved.
[10:8]	Max clock latency	Set from **MAXCLKLATENCY[2:0]**.
[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
```

4.3.68 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 processor 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 4-26 on page 4-31.

Figure 4-67 on page 4-127 shows the Configuration Base Address Register bit assignments.

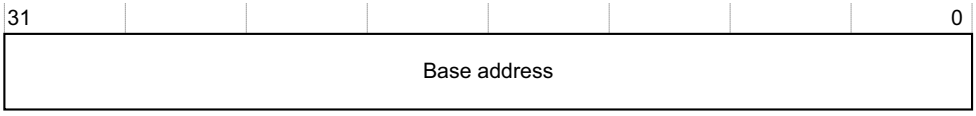


Figure 4-67 Configuration Base Address Register bit assignments

To access the Configuration Base Address Register, use:

```
MRC p15,0,<Rd>,c15,c4,0; Read Configuration Base Address Register
MCR p15,0,<Rd>,c15,c4,0; Write Configuration Base Address Register
```

4.4 CP14 Jazelle registers

In the Cortex-A9 implementation of the Jazelle Extension:

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

Table 4-87 shows the CP14 Jazelle registers. All Jazelle registers are 32 bits wide.

Table 4-87 CP14 Jazelle registers summary

CRn	Op1	CRm	Op2	Name	Type	Reset	Page
0	7	0	0	JIDR	RW[a]	`0xF4100168`	page 4-129
7	1	0	0	JOSCR	RW	-	page 4-130
7	2	0	0	JMCR	RW	-	page 4-132
7	3	0	0	Jazelle Parameters Register	RW	-	page 4-134
7	4	0	0	Jazelle Configurable Opcode Translation Table Register	WO	-	page 4-135

a. See *Write operation of the JIDR* on page 4-130 for the effect of a write operation

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

4.5 CP14 Jazelle register descriptions

The following sections describe the CP14 Jazelle DBX registers arranged in numerical order, as shown in Table 4-87 on page 4-128:

- *Jazelle Identity and Miscellaneous Functions Register*
- *Jazelle Operating System Control Register* on page 4-130
- *Jazelle Main Configuration Register* on page 4-132
- *Jazelle Parameters Register* on page 4-134
- *Jazelle Configurable Opcode Translation Table Register* on page 4-135.

4.5.1 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 4-130.

Configurations Available in all configurations.

Attributes See the register summary in Table 4-87 on page 4-128.

Figure 4-68 shows the JIDR bit assignments.

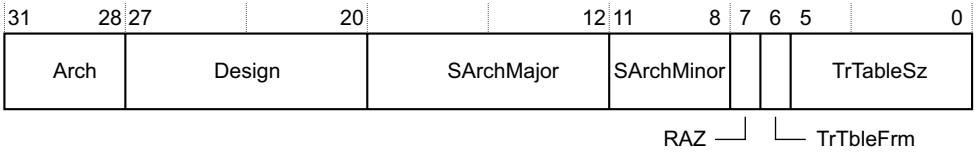


Figure 4-68 JIDR bit assignment

Table 4-88 shows the JIDR bit assignments.

Table 4-88 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	TrTblFrm indicates the format of the Jazelle Configurable Opcode Translation Table Register.
[5:0]	TrTbleSz	TrTblFrm indicates the format 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 4-135.

4.5.2 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 4-87 on page 4-128.

Figure 4-69 shows the JOSCR bit assignments.

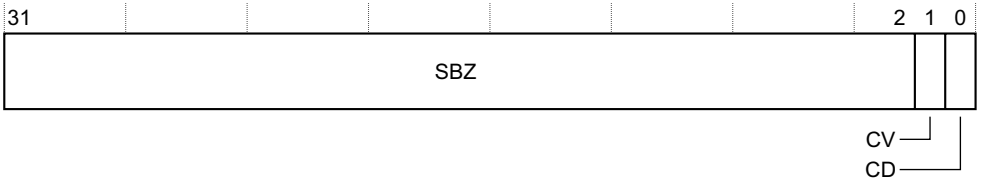


Figure 4-69 JOSCR bit assignments

Table 4-89 shows the JOSCR bit assignments.

Table 4-89 JOSCR bit assignments

Bits	Name	Description
[31:2]	-	SBZ
[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
```

4.5.3 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 4-87 on page 4-128.

Figure 4-70 shows the JMCR bit assignments.

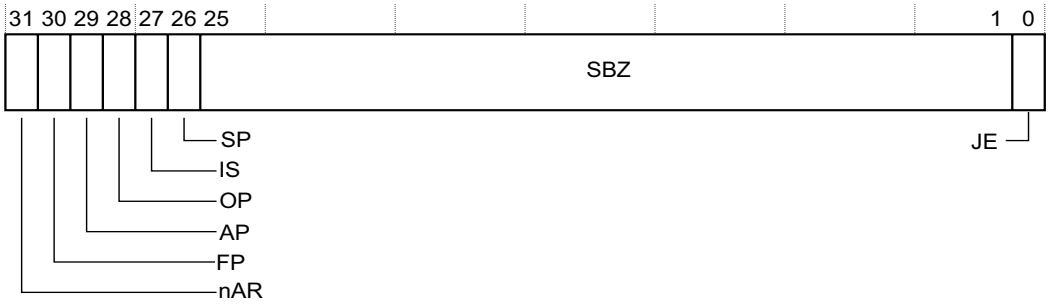


Figure 4-70 JMCR bit assignments

Table 4-90 shows the JMCR bit assignments.

Table 4-90 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 4-90 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]	SBZ	-
[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
```

4.5.4 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 4-87 on page 4-128.

Figure 4-71 shows the Jazelle Parameters Register bit assignments.

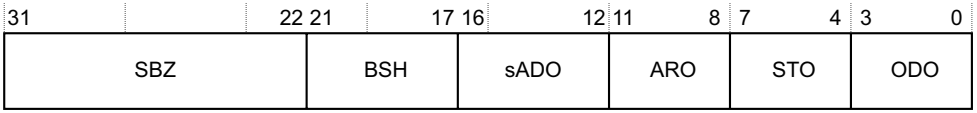


Figure 4-71 Jazelle Parameters Register bit assignments

Table 4-91 shows the Jazelle Parameters Register bit assignments.

Table 4-91 Jazelle Parameters Register bit assignments

Bits	Name	Description
[31:22]	-	SBZ
[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
```

4.5.5 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 4-87 on page 4-128.

Figure 4-72 shows the Jazelle Configurable Opcode Translation Table Register bit assignments.

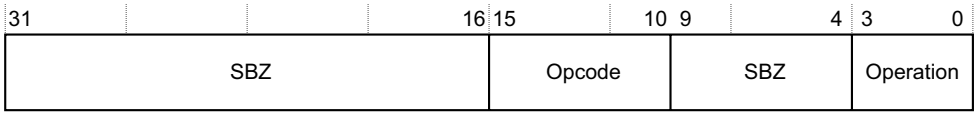


Figure 4-72 Jazelle Configurable Opcode Translation Table Register bit assignments

Table 4-92 shows the Jazelle Configurable Opcode Translation Table Register bit assignments.

Table 4-92 Jazelle Configurable Opcode Translation Table Register bit assignments

Bits	Name	Description
[31:16]	-	SBZ
[15:10]	Opcode	Contains the bottom bits of the configurable opcode.
[9:4]	-	SBZ
[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.

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

Four page sizes are supported:

- 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 are implemented in logic, providing 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. This is necessary because page table attributes such as ASIDs are not stored in micro TLBs.

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 RAM 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 4-113 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 Register 0* on page 4-75 and *Translation Table Base Register 1* on page 4-77. 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.

   See *c10, Memory region remap* on page 4-15.
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. See *Secure Configuration Register* on page 4-68 for more information.

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 Coprocessor* 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.

5.6 MMU software-accessible registers

The system control coprocessor registers, CP15, in conjunction with page table descriptors stored in memory, control the MMU as shown in Table 5-1.

You can access all the registers with instructions of the form:

```
MRC p15, 0,<Rd>, <CRn>, <CRm>, <Opcode_2>
```

```
MCR p15, 0,<Rd>, <CRn>, <CRm>, <Opcode_2>
```

CRn is the system control coprocessor register. Unless specified otherwise, CRm and Opcode_2 Should Be Zero.

Table 5-1 CP15 register functions

Register	Cross reference
TLB Type Register	*Cache Type Register* on page 4-34.
Control Register	*System Control Register* on page 4-59
Non-secure Access Control Register	*Non-secure Access Control Register* on page 4-71
Translation Table Base Register 0	*Translation Table Base Register 0* on page 4-75
Translation Table Base Register 1	*Translation Table Base Register 1* on page 4-77
Translation Table Base Control Register	*Translation Table Base Control Register* on page 4-79
Domain Access Control Register	*Domain Access Control Register* on page 4-82
DFSR	*Instruction Fault Status Register* on page 4-86
IFSR	*Instruction Fault Status Register* on page 4-86.
DFAR	*Data Fault Status Register* on page 4-83
IFAR	*Instruction Fault Address Register* on page 4-89
TLB operations	*c8, TLB Operations Register* on page 4-13.
TLB Lockdown Registers	*TLB Lockdown Register* on page 4-113
Primary Region Remap Register	*c10, Memory region remap* on page 4-15
Normal Memory Remap Register	
ContextID Register	*Context ID Register* on page 4-124.

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 4-59.
- 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. If an instruction loop is small enough to fit in four consecutive 16 byte blocks aligned on 16 bytes, then instruction cache accesses are turned off, reducing power consumption.

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 Cortex-A9 cache policies

The Cortex-A9 processor implements a Write-Back Write-Allocate cache allocation policy. Table 6-1 shows the cache policies and how they are implemented in the Cortex-A9 processor.

Table 6-1 Cortex-A9 cache policies

Cache policy	Implemented as
Normal Non-cacheable (NC)	Normal Non-cacheable
Normal Write-Through (WT)	Normal Non-cacheable
Normal Write-Back (WB)	Write-Back, Write-Allocate

The default Write-Back Write-Allocate cache allocation policy can change dynamically into Write-Back no Write-Allocate depending on the access pattern produced by the processor. This allocation policy switch is done automatically by the cache controller and does not require any software hint.

6.3 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.

6.4 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.

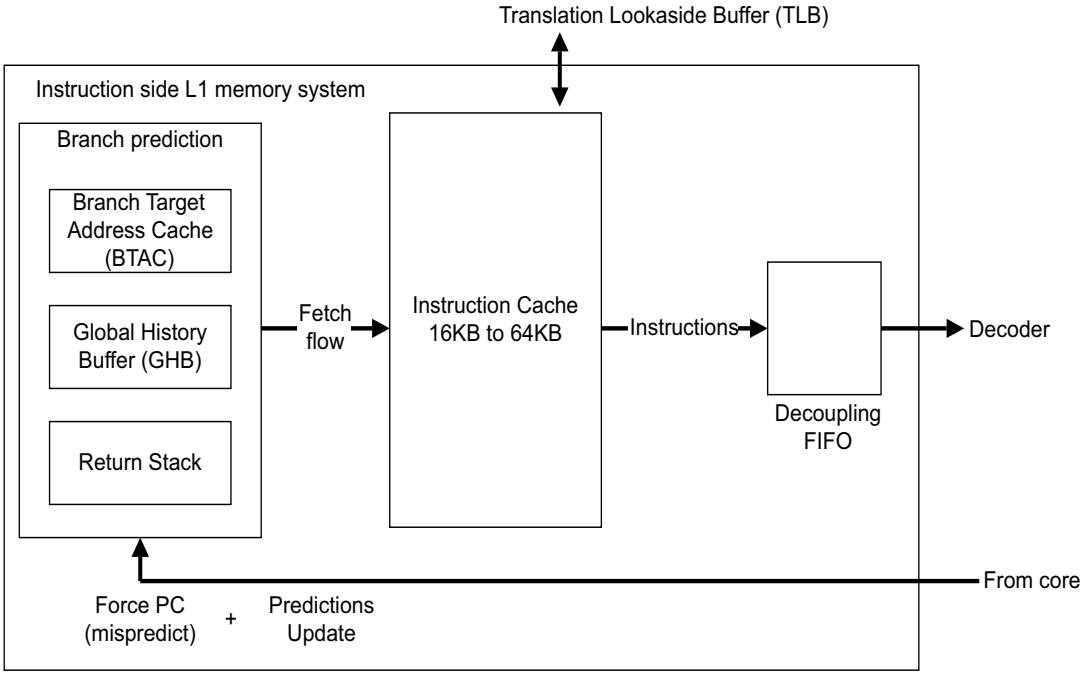


Figure 6-1 Branch prediction and instruction cache controller

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.

6.4.1 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 4-59. 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.

6.4.2 Program flow prediction

The following sections describe program flow prediction:

- *Predicted and non-predicted instructions*
- *Thumb state conditional branches* on page 6-8
- *Return stack predictions* on page 6-8.

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.

6.5 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.

6.5.1 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 8-12 on page 8-32

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.

6.5.2 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.

6.6 Data prefetching

This section describes:

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

6.6.1 The PLD instruction

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

6.6.2 Data prefetching and monitoring

The Cortex-A9 data cache implements an automatic prefetcher that monitors cache misses done by the processor. This unit can monitor and prefetch two independent data streams. It can be activated in software using a CP15 Auxiliary Control Register bit. See *Auxiliary Control Register* on page 4-63.

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.

6.7 Parity error support

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

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

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

Figure 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.

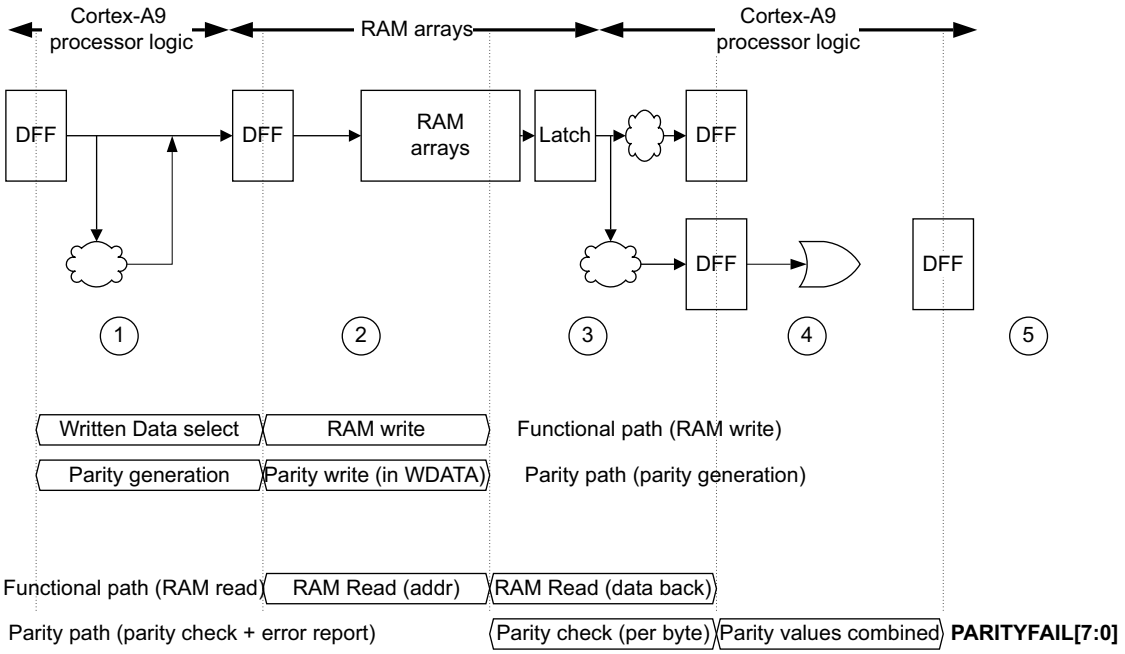


Figure 6-2 Parity support

The output signals **PARITYFAIL[7:0]** report parity errors. Typically, **PARITYFAIL[7:0]** reports parity errors 3 clock cycles after the corresponding RAM read. You can tie unused bits of **PARITYFAIL[7:0]** HIGH. See *Parity signal* on page A-20.

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.

6.7.1 GHB and BTAC data corruption

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

Chapter 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]	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

Instruction side read bus, ARUSERM1[6:0]

Table 7-4 shows the bit encodings for **ARUSERM0[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 ARUSERM1[8:0] encodings

Bits	Name	Description
[8]	Reserved	b0
[7]	Reserved	b0
[6]	Clean eviction	-
[5]	L1 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 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. See *Auxiliary Control Register* on page 4-63.

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, Cortex-A9 MPCore only*
- *Early BRESP*
- *Prefetch hint to the L2 memory interface, Cortex-A9 MPCore only*.

7.2.1 Prefetch hint to the L2 memory interface, Cortex-A9 MPCore only

When this features is enabled, the automatic data prefetch engine in the Cortex-A9 processor generates prefetch hint requests to the L2 memory interface when it detects regular fetch patterns on coherent memory accesses.

This feature can optimize the performance of the processor, but it generates non-AXI compliant requests. These prefetch hint requests are AXI read requests which do not expect any data back. The requests are marked as prefetch hints by setting the associated **ARUSER[5]** bit.

Setting Bit[1] of the ACTRL enables this feature. No additional programming of the PL310 cache controller 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 PL310 cache controller must also be programmed to support this feature. See *PL310 Cache Controller TRM*.

7.2.3 Write full line of zero

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 zero writes by having the associated **AWUSER[7]** bit set.

Setting bit[3] of the ACTLR enables this feature.

The PL310 cache controller must also be programmed to support this feature, prior to enabling the feature in the Cortex-A9 processor. See *PL310 Cache Controller 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 **APROT** values.

Table 7-6 Cortex-A9 mode and APROT values

Processor mode	Type of access	Value of APROT
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
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:

8.1 About debug systems

The processor forms one component of a debug system. Figure 8-1shows a typical system.

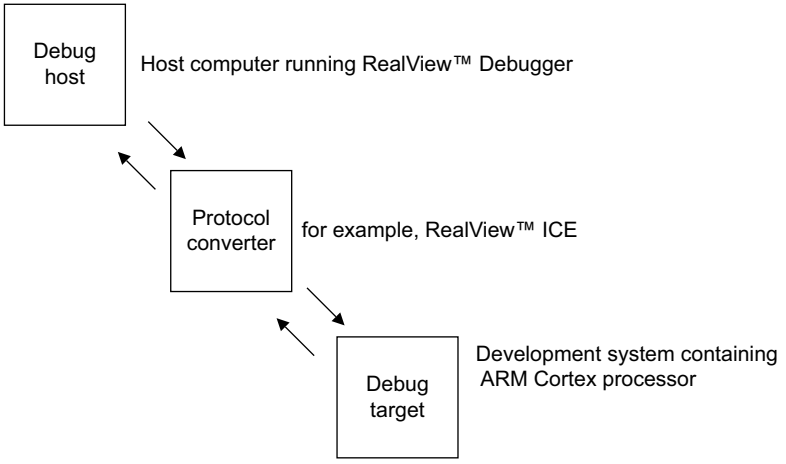


Figure 8-1 Typical debug system

This typical system has a:

- debug host
- protocol converter
- debug target.

8.1.1 Debug host

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

8.1.2 Protocol converter

The debug host sends messages to the debug target using an interface such as Ethernet. However, the debug target typically implements a different interface protocol. A device such as RealView ICE is required to convert between the two protocols.

8.1.3 Debug target

The debug target is the lowest level of the system. An example of a debug target is a development system with a test chip or a silicon part with a processor. The debug target implements system support for the protocol converter to access the Cortex-A9 Debug Unit using the APB slave port.

The APB slave port is compliant with version 3 of the ARM AMBA™ *Advanced Peripheral Bus* specification. This APB slave interface supports 32-bits wide data, stalls, slave-generated aborts, and eleven address bits [12:2] mapping 2x4KB of memory. An extra **PADDR31** signal indicates to the Cortex-A9 Debug Unit the source of the access. The debug bus accesses both debug and PMU CoreSight components inside the Cortex-A9. See *External Debug interface* on page A-23 for a complete list of the debug APB signals.

8.1.4 About the debug unit

The processor debug unit assists in debugging software running on the processor. You can use the processor debug unit, in combination with a software debugger program, to debug:

- application software
- operating systems
- hardware systems based on an ARM processor.

The debug unit enables you to:

- stop program execution
- examine and alter processor and coprocessor state
- examine and alter memory and input/output peripheral state
- restart the processor core.

You can debug software running on the processor in the following ways:

- Halting debug-mode debugging
- Monitor debug-mode debugging
- trace debugging, see *Interfaces* on page 2-4.

8.2 Debugging modes

The processor implements these types of debug:

Invasive debug

Invasive debug is defined as a debug process where you can control and observe the processor. Most debug features in this chapter are considered invasive debug because they enable you to halt the processor and modify its state.

SPIDEN controls invasive debug permissions.

Noninvasive debug

Noninvasive debug is defined as a debug process where you can observe the processor but not control it. The PTM interface and the performance monitor registers are features of noninvasive debug.

See *Interfaces* on page 2-4 for information on the PTM interface.

See *System performance monitor registers* on page 4-11 for information on performance monitor registers.

SPNIDEN controls noninvasive debug permissions.

The following sections describe:

- *Halting debug-mode debugging*
- *Monitor debug mode debugging* on page 8-5
- *Security extensions and debugging* on page 8-5.

8.2.1 Halting debug-mode debugging

When the processor debug unit is in Halting debug-mode, the processor halts when a debug event, such as a breakpoint, occurs. When the processor is halted, an external debugger can examine and modify the processor state using the APB slave port. This debug mode is invasive to program execution.

Note

The Cortex-A9 processor does not support Secure User Halting Debug-Mode unless secure privilege debugging is also enabled. You can bypass this restriction by setting the external **SPIDEN** pin HIGH.

8.2.2 Monitor debug mode debugging

When the processor debug unit is in Monitor debug-mode and a debug event occurs, the processor takes a debug exception instead of halting. A special piece of software, a monitor target, can then take control to examine or alter the processor state. Monitor debug-mode is essential in real-time systems where the processor cannot be halted to collect debug information. Examples of these systems are engine controllers and servo mechanisms in hard drive controllers that cannot stop the code without physically damaging the components.

8.2.3 Security extensions and debugging

To prevent access to secure system software or data while still permitting Non-secure state and optionally secure User mode to be debugged, you can set debug to one of three levels:

- Non-secure state only
- Non-secure state and Secure User mode only. Only Monitor Mode debugging is supported for secure User mode.
- any Secure or Non-secure state.

The **SPIDEN** and **SPNIDEN** signals, and the two bits, **SUIDEN** and **SUNIDEN**, in the Secure Debug Enable Register in CP15 coprocessor control the secure debug permissions.

See *Secure Debug Enable Register* on page 4-70, *Authentication signals* on page 8-50, and *Changing the authentication signals* on page 8-51 for details.

8.3 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
- system coherency events.

See *Event Selection Register* on page 4-101 for a description of the events.

8.3.1 Breakpoints and watchpoints

There are:

- six breakpoints, two with Context ID comparison capability, BRP4 and BRP5. See *Breakpoint Value Registers* on page 8-26 and *Breakpoint Control Registers* on page 8-28.
- 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. See *CP14 c1, Debug Status and Control Register (DBGDSCR)* on page 8-16

  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 8-31 and *Watchpoint Control Registers* on page 8-32.

8.3.2 Asynchronous aborts

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

8.3.3 Processor interfaces

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

Debug registers

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

You can access the Cortex-A9 debug register map using the APB slave port. See *CTI signals* on page A-25 and *APB interface signals* on page A-24

Performance monitor

This interface is CP15 based and memory-mapped. See *System performance monitor registers* on page 4-11 for information on performance monitor registers.

Trace registers

This interface is memory-mapped.

The Cortex-A9 processor implements PFT, an instruction-only trace architecture. See *Interfaces* on page 2-4.

8.3.4 Effects of resets on debug registers

nCPURESET

The **nCPURESET** signal is the main Cortex-A9 processor reset that initializes the Cortex-A9 processor logic. It has no effect on the debug logic.

nDBGRESET

The **nDBGRESET** signal is the debug logic reset signal. A power-on reset asserts nCPURESET and nDBGRESET.

nDERESET resets Data Engine 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.

Figure 8-2 on page 8-8 shows the Cortex-A9 connections specific to debug request and restart.

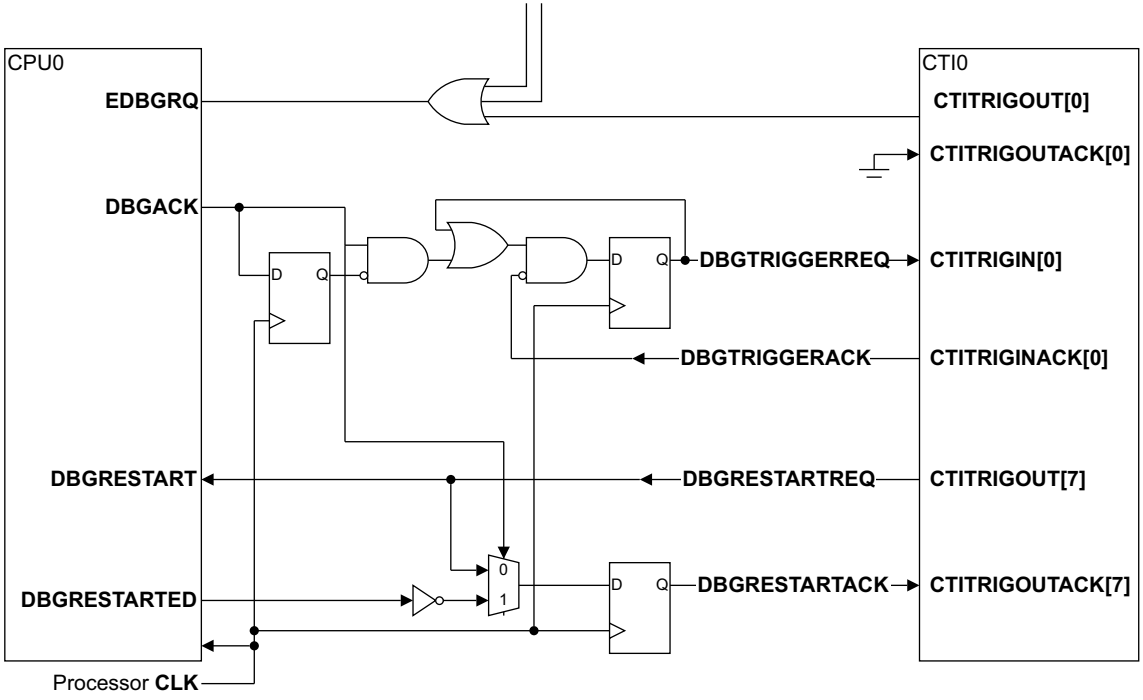


Figure 8-2 Debug request restart-specific connections

8.4 About the Cortex-A9 debug register interface

The debug register 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 8-3 shows the Cortex-A9 debug registers interface.

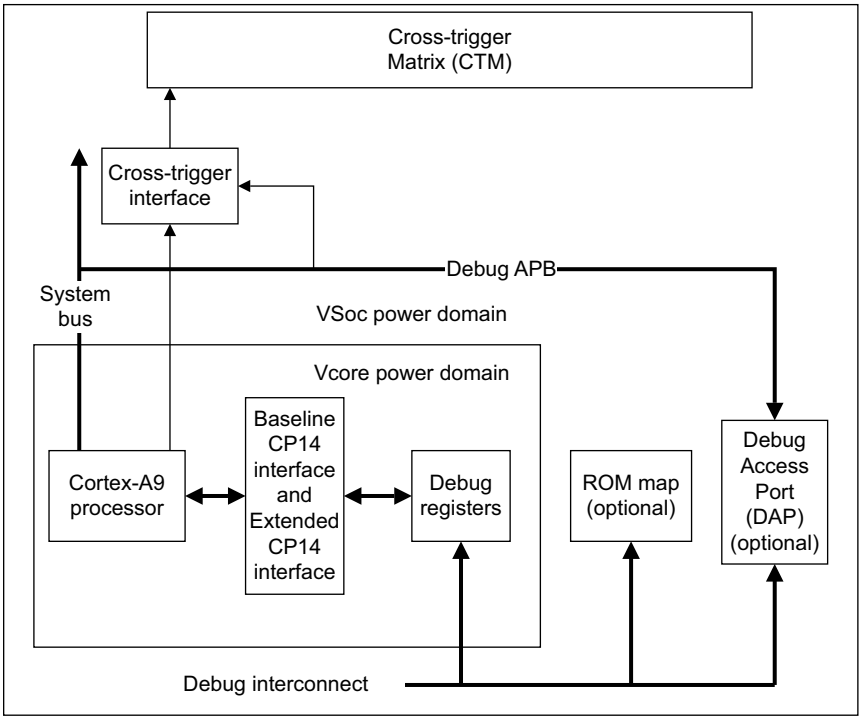


Figure 8-3 Debug registers interface

8.4.1 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 8-1 shows the CP14 interface registers. All other registers are described in the *ARM Architecture Reference Manual*.

Table 8-1 CP14 interface registers

Register number	Offset	CP14 instruction	Access	Register name	Description
0	0x000	0 c0 c0 0	RO	DBGDIDR[a]	*CP14 c0, Debug ID Register (DBGDIDR)* on page 8-14[b]
-	-	0 c1 c0 0	RO	DBGDRAR[a]	-
-	-	0 c2 c0 0	RO	DBGDSAR[a]	-
-	-	0 c0 c1 0	RO	DBGDSCR-int[ab]	*CP14 c1, Debug Status and Control Register (DBGDSCR)* on page 8-16[a]
-	-	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 8-25
11	0x02C	0 c0 c11 0	RAZ/WI	DBGDSMCR	Not implemented

Table 8-1 CP14 interface registers (continued)

Register number	Offset	CP14 instruction	Access	Register name	Description
12-31	-	-	-	Reserved	-
32	0x080	0 c0 c0 2	RW	DBGDTRRX -ext	-
33	0x084	0 c0 c1 2	WO	DBGITR	-
33	0x084	0 c0 c1 2	RO	DBGPCSR	*Program Counter Sampling Register (DBGPCSR)* on page 8-24
34	0x088	0 c0 c2 2	RW	DBGDSCR-ext	*CP14 c1, Debug Status and Control Register (DBGDSCR)* on page 8-16
35	0x08C	0 c0 c3 2	RW	DBGDTRTX-ext	-
36	0x090	0 c0 c4 2	WO	DBGDRCR	*Debug Run Control Register (DBGDRCR)* on page 8-25
37-63	-	-	-	Reserved	-
64-79	0x100-0x13C	0 c0 c0-15 4	RW	DBGBVRn	*Breakpoint Value Registers* on page 8-26
80-95	0x140-0x17C	0 c0 c0-15 5	RW	DBGBCRn	*Breakpoint Control Registers* on page 8-28
96-111	0x180-0x1BC	0 c0 c0-15 6	RW	DBGWVRn	*Watchpoint Value Registers* on page 8-31
112-127	0x1C0-0x1FC	0 c0 c0-15 7	RW	DBGWCRn	*Watchpoint Control Registers* on page 8-32
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	-

Table 8-1 CP14 interface registers (continued)

Register number	Offset	CP14 instruction	Access	Register name	Description
196	0x310	0 c1 c4 4	RW	DBGPRCR	*Device Power-down and Reset Control Register (DBGPRCR)* on page 8-34
197	0x314	0 c1 c5 4	RW	DBGPRSR	*Device Power-down and Reset Status Register (DBGPRSR)* on page 8-35
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	-
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 (DBGITCTRL)* on page 8-39
961-999	0xF04-0xF9C	-	-	-	-
1000	0xFA0	0 c7 c8 6	RW	DBGCLAIMSET	*Claim Tag Set Register (DBGCLAIMSET)* on page 8-39
1001	0xFA4	0 c7 c9 6	RW	DBGCLAIMCLR	*Claim Tag Clear Register (DBGCLAIMCLR)* on page 8-40
1002-1003	-	-	-	Reserved	-
1004	0xFB0	0 c7 c12 6	WO	DBGLAR	*Lock Access Register (DBGLAR)* on page 8-41
1005	0xFB4	0 c7 c13 6	RO	DBGLSR	*Lock Status Register (DBGLSR)* on page 8-41

Table 8-1 CP14 interface registers (continued)

Register number	Offset	CP14 instruction	Access	Register name	Description
1006	0xFB8	0 c7 c14 6	RO	DBGAUTHSTATUS	*Authentication Status Register (DBGAUTHSTATUS)* on page 8-42
1007-1009	-	-	-	Reserved	-
1010	0xFC8	0 c7 c2 7	RAZ	DBGDEVID	-
1011	0xFCC	0 c7 c3 7	RO	DBGDEVTYPE	*Device Type Register (DBGDEVTYPE)* on page 8-43
1012-1016	0xFD0-0xFEC	0 c7 c4-8 7	RO	PERIPHERALID	*Identification Registers* on page 8-44
1017-1019	-	-	-	Reserved	-
1020-1023	0xFF0-0xFFC	0 c7 c12-15 7	RO	COMPONENTID	*Identification Registers* on page 8-44

a. Baseline CP14 interface. This register also has an external view through the memory-mapped interface and the CP14 interface. See *CP14 c1, Debug Status and Control Register (DBGDSCR)* on page 8-16.
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. See *Performance Monitor Control Register* on page 4-92. Reads from the extended CP14 interface return zero.

8.5 Debug register descriptions

This section describes the debug registers.

8.5.1 CP14 c0, Debug ID Register (DBGDIDR)

The DBGDIDR is a read-only register that identifies the debug architecture version and specifies the number of debug resources that the Cortex-A9 processor implements.

The Debug ID Register is:

- in CP14 c0
- a read-only register
- accessible in User and privileged modes.

Note

All Baseline CP14 registers are accessible in User mode only if DBGDSCR.UDCCDdis=0.

Figure 8-4 shows the bit arrangement of the DBGID Register.

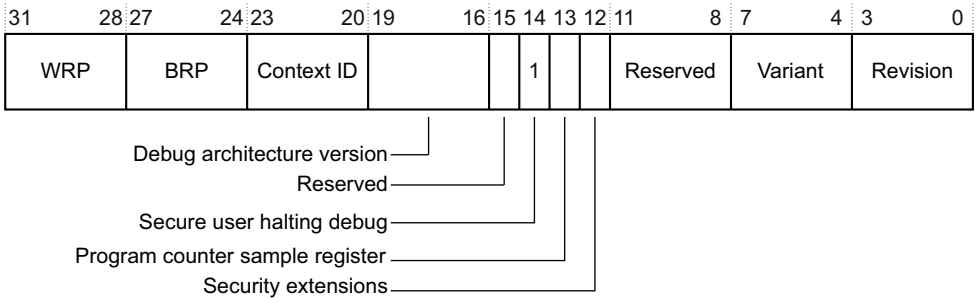


Figure 8-4 Debug ID Register bit assignments

Table 8-2 shows how the bit values correspond with the Debug ID Register functions.

Table 8-2 Debug ID Register bit assignments

Bits	Name	Description
[31:28]	WRP	Number of Watchpoint Register Pairs: For the Cortex-A9 processor, this field reads b0001 to indicate two WRPs are implemented.
[27:24]	BRP	Number of Breakpoint Register Pairs: For the Cortex-A9 processor, this field reads b0101 to indicate six BRPs are implemented.
[23:20]	Context	Number of Breakpoint Register Pairs with context ID comparison capability: For the Cortex-A9 processor, this field reads b0001 to indicate two BRPs have context ID capability.
[19:16]	Debug architecture version	Debug architecture version: b0011 = ARMv7 Debug with Extended CP14 interface implemented.
[15]	-	Reserved.
[14]	Secure User halting debug-mode	For the Cortex-A9 processor, this field reads as b1 to indicate that Secure User halting debug-mode is not supported.
[13]	Program Counter Sampling Register.	Program Counter Sample, DBGPCSR, register. For the Cortex-A9 processor, this field reads as b1 to indicate that DBGPCSR is implemented.
[12]	Security extensions	Security extensions bit: For the Cortex-A9 processor, this field reads 1 to indicate that the debug security extensions are implemented.
[11:8]	-	RAZ.
[7:4]	Variant	Implementation-defined variant number. This number is incremented on functional changes. The value matches bits [23:20] of the ID Code Register in CP15 c0.
[3:0]	Revision	Implementation-defined revision number. This number is incremented on bug fixes. The value matches bits of the ID Code Register in CP15 c0.

The values of the following fields of the Debug ID Register agree with the values in CP15 c0, Main ID Register:

- DIDR is the same as CP15 c0 bits
- DIDR[7:4] is the same as CP15 c0 bits [23:20].

See Figure 4-14 on page 4-33 for a description of CP15 c0, Main ID Register.

To use the Debug ID Register, read CP14 c0 with:

- Opcode_1 set to 0
- CRn set to c0
- CRm set to c0
- Opcode_2 set to 0.

For example:

```
MRC p14, 0, <Rd>, c0, c0, 0 ; Read Debug ID Register
```

8.5.2 CP14 c1, Debug Status and Control Register (DBGDSCR)

The DBGDSCR contains status and control information about the debug unit. Figure 8-5 on page 8-17 shows the bit arrangement of the DBGDSCR.

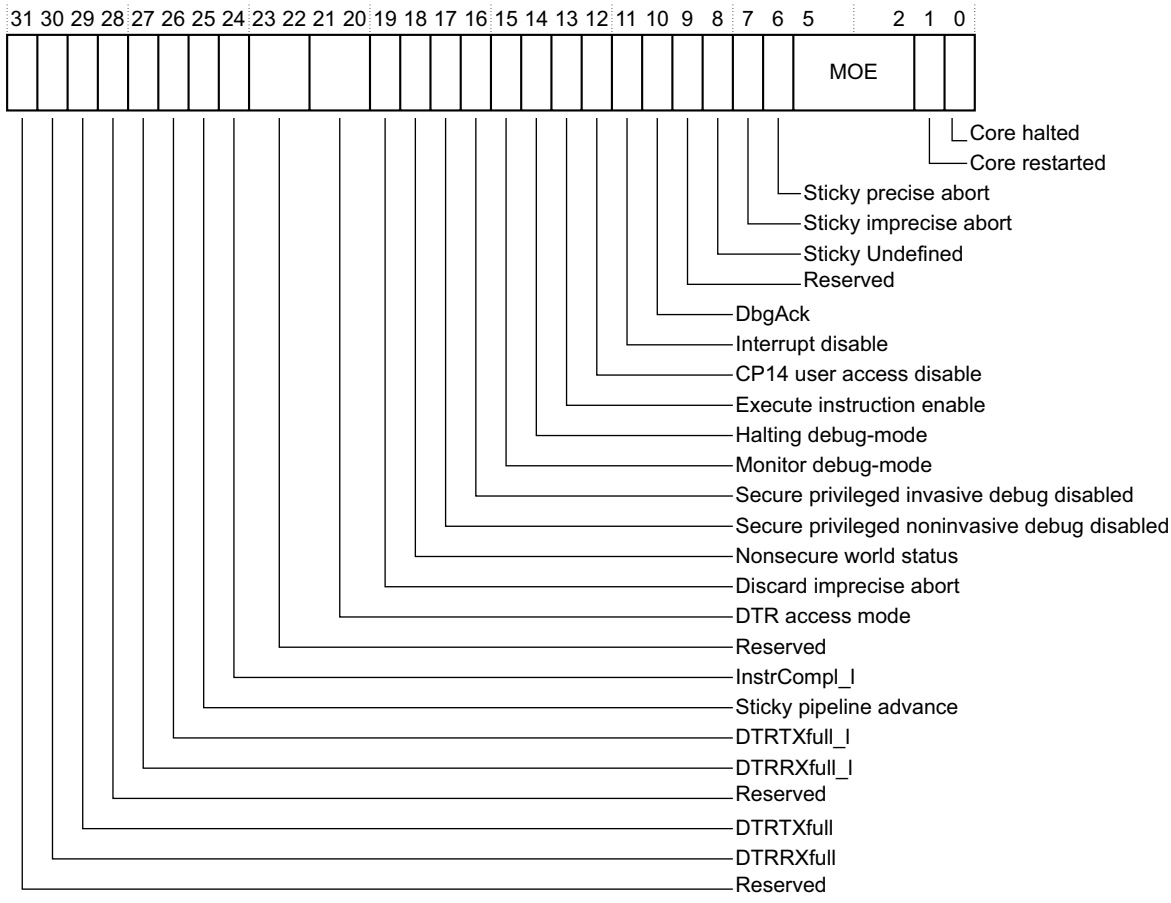


Figure 8-5 Debug Status and Control Register bit assignments

Table 8-3 shows how the bit values correspond with the Debug Status and Control Register functions.

Table 8-3 Debug Status and Control Register bit assignments

Bits	Name	Description
[31]	-	RAZ on reads, SBZP on writes.
[30]	DTRRXfull	The DTRRXfull flag: 0 = DTRRX empty, reset value 1 = DTRRX full. When set, this flag indicates that there is data available in the Receive Data Transfer Register, DTRRX. It is automatically set on writes to the DTRRX by the debugger, and is cleared when the processor reads the CP14 DTR. If the flag is not set, the DTRRX returns an Unpredictable value.
[29]	DTRTXfull	The DTRTXfull flag: 0 = DTRTX empty, reset value 1 = DTRTX full. When clear, this flag indicates that the Transmit Data Transfer Register, DTRTX is ready for data write. It is automatically cleared on reads of the DTRTX by the debugger, and is set when the processor writes to the CP14 DTR. If this bit is set and the core attempts to write to the DTRTX, the register contents are overwritten and the DTRTXfull flag remains set.
[28]	-	RAZ on reads, SBZP on writes.
[27]	DTRRXfull_l	The latched DTRRXfull flag. This flag is read in one of the following ways: • using `CP14` instructions • using the DSCR memory address. `CP14` instruction returns an Unpredictable value for this bit. DSCR memory address returns the same value as DTRRXfull. If a write to the DTRRX APB address succeeds, DTRRXfull_l is set to 1.
[26]	DTRTXfull_l	The latched DTRTXfull flag. This flag is read in one of the following ways: • using `CP14` instruction • using the DSCR memory address. `CP14` instruction returns an Unpredictable value for this bit. DSCR memory address returns the same value as DTRTXfull. If a read to the DTRTX APB address succeeds, DTRTXfull_l is cleared.

Table 8-3 Debug Status and Control Register bit assignments (continued)

Bits	Name	Description
[25]	Sticky pipeline advance	Sticky pipeline advance bit. This bit enables the debugger to detect whether the processor is idle. In some situations, this might mean that the system bus port is deadlocked. This bit is set to 1 every time the processor pipeline retires one instruction. A write to DRCR[3] clears this bit. See *Debug Run Control Register (DBGDRCR)* on page 8-25. 0 = no instruction has completed execution since the last time this bit was cleared, reset value 1 = an instruction has completed execution since the last time this bit was cleared.
[24]	InstrCompl_l	The latched InstrCompl flag. This flag is read in one of the following ways: • using CP14 instruction • using the DSCR memory address. CP14 instruction returns an Unpredictable value for this bit. DSCR memory address returns the same value as InstrCompl. If a write to the ITR APB address succeeds while in Stall or Nonblocking mode, InstrCompl_l and InstrCompl are cleared. If a write to the DTRRX APB address or a read to the DTRTX APB address succeeds while in Fast mode, InstrCompl_l and InstrCompl are cleared. InstrCompl is the instruction complete bit. This internal flag determines whether the processor has completed execution of an instruction issued through the APB port. 0 = the processor is currently executing an instruction fetched from the ITR Register, reset value 1 = the processor is not currently executing an instruction fetched from the ITR Register.
[23:22]	-	RAZ on reads, SBZP on writes.
[21:20]	DTR access mode	DTR access mode. This is a read and write field. You can use this field to optimize DTR traffic between a debugger and the processor: b00 = Nonblocking mode, reset value b01 = Stall mode b10 = Fast mode b11 = reserved. **Note** • This field only affects the behavior of DSCR, DTR, and ITR accesses through the APB port, and not through CP14 debug instructions. • Nonblocking mode is the default setting. Improper use of the other modes might result in the debug access bus becoming jammed. See *DTR access mode* on page 8-23 for more information.

Table 8-3 Debug Status and Control Register bit assignments (continued)

Bits	Name	Description
[19]	Discard asynchronous abort	Discard asynchronous abort. This read-only bit is set while the processor is in debug state and is cleared on exit from debug state. While this bit is set, the processor does not record asynchronous Data Aborts. However, the sticky asynchronous Data Abort bit is set to 1. 0 = asynchronous Data Aborts not discarded, reset value 1 = asynchronous Data Aborts discarded.
[18][a]	Non-secure state status	Non-secure state status bit: 0 = the processor is in Secure state or the processor is in Monitor mode 1 = the processor is in Non-secure state and is not in Monitor mode.
[17][a]	Secure privileged noninvasive debug disabled	Secure privileged noninvasive debug disabled: 0 = ((**NIDEN** || **DBGEN**) && (**SPNIDEN** || **SPIDEN**)) is HIGH 1 = ((**NIDEN** || **DBGEN**) && (**SPNIDEN** || **SPIDEN**)) is LOW. This value is the inverse of bit [6] of the Authentication Status Register. See *Authentication Status Register (DBGAUTHSTATUS)* on page 8-42.
[16][a]	Secure privileged invasive debug disabled	Secure privileged invasive debug disabled: 0 = (**DBGEN** && **SPIDEN**) is HIGH 1 = (**DBGEN** && **SPIDEN**) is LOW. This value is the inverse of bit [4] of the Authentication Status Register. See *Authentication Status Register (DBGAUTHSTATUS)* on page 8-42.
[15]	Monitor debug-mode	The Monitor debug-mode enable bit. This is a read and write bit. 0 = Monitor debug-mode disabled, reset value 1 = Monitor debug-mode enabled. If Halting debug-mode is enabled, bit [14] is set, then the processor is in Halting debug-mode regardless of the value of bit [15]. If the external interface input **DBGEN** is LOW, DSCR[15] reads as 0. If **DBGEN** is HIGH, then the read value reverts to the programmed value.
[14]	Halting debug-mode	The Halting debug-mode enable bit. This is a read and write bit. 0 = Halting debug-mode disabled, reset value 1 = Halting debug-mode enabled. If the external interface input **DBGEN** is LOW, DSCR[14] reads as 0. If **DBGEN** is HIGH, then the read value reverts to the programmed value.
[13]	Execute instruction enable	Execute ARM instruction enable bit. This is a read and write bit. 0 = disabled, reset value 1 = enabled. If this bit is set and an ITR write succeeds, the processor fetches an instruction from the ITR for execution. If this bit is set to 1 when the processor is not in debug state, the behavior of the processor is Unpredictable.

Table 8-3 Debug Status and Control Register bit assignments (continued)

Bits	Name	Description
[12]	CP14 user access disable	CP14 debug user access disable control bit. This is a read and write bit. 0 = CP14 debug user access enable, reset value 1 = CP14 debug user access disable. If this bit is set and a User mode process tries to access any CP14 debug registers, the Undefined instruction exception is taken.
[11]	Interrupt disable	Interrupts disable bit. This is a read and write bit. 0 = interrupts enabled, reset value 1 = interrupts disabled. If this bit is set, the **IRQ** and **FIQ** input signals are disabled. The external debugger can set this bit before it executes code in normal state as part of the debugging process. If this bit is set to 1, an interrupt does not take control of the program flow. For example, the debugger might use this bit to execute an OS service routine to bring a page from disk into memory. It might be undesirable to service any interrupt during the routine execution.
[10]	DbgAck	Debug Acknowledge bit. This is a read and write bit. If this bit is set to 1, both the **DBGACK** and **DBGTRIGGER** output signals are forced HIGH, regardless of the processor state. The external debugger can use this bit if it wants the system to behave as if the processor is in debug state. Some systems rely on **DBGACK** to determine whether the application or debugger generates the data accesses. The reset value is 0.
[9]	-	RAZ on reads, SBZP on writes.
[8]	Sticky Undefined	Sticky Undefined bit: 0 = No Undefined exception occurred in debug state since the last time this bit was cleared. This is the reset value. 1 = An Undefined exception has occurred while in debug state since the last time this bit was cleared. This flag detects Undefined exceptions generated by instructions issued to the processor through the ITR. This bit is set to 1 when an Undefined instruction exception occurs while the processor is in debug state. Writing a 1 to DRCR[2] clears this bit. See *Debug Run Control Register (DBGDRCR)* on page 8-25.
[7]	Sticky asynchronous abort	Sticky asynchronous Data Abort bit: 0 = no asynchronous Data Aborts occurred since the last time this bit was cleared, reset value 1 = an asynchronous Data Abort occurred since the last time this bit was cleared. This flag detects asynchronous Data Aborts triggered by instructions issued to the processor through the ITR. This bit is set to 1 when an asynchronous Data Abort occurs while the processor is in debug state. Writing a 1 to DRCR[2] clears this bit. See *Debug Run Control Register (DBGDRCR)* on page 8-25.

Table 8-3 Debug Status and Control Register bit assignments (continued)

Bits	Name	Description
[6]	Sticky synchronous abort	Sticky synchronous Data Abort bit: 0 = no synchronous Data Abort occurred since the last time this bit was cleared, reset value 1 = a synchronous Data Abort occurred since the last time this bit was cleared. This flag detects synchronous Data Aborts generated by instructions issued to the processor through the ITR. This bit is set to 1 when a synchronous Data Abort occurs while the processor is in debug state. Writing a 1 to DRCR[2] clears this bit. See *Debug Run Control Register (DBGDRCR)* on page 8-25.
[5:2]	MOE	MOE, Method of entry bits. This is a read and write field. b0000 = a DRCR[0] halting debug event occurred, reset value b0001 = a breakpoint occurred b0010 = not supported b0011 = a `BKPT` instruction occurred b0100 = an **EDBGRQ** halting debug event occurred b0101 = vector catch b1010 = OS synchronous watchpoint other = reserved. These bits are set to indicate any of: • the cause of a debug exception • the cause for entering debug state. A Prefetch Abort or Data Abort handler must check the value of the CP15 Fault Status Register to determine whether a debug exception occurred and then use these bits to determine the specific debug event.
[1][a]	Core restarted	Core restarted bit: 0 = The processor is exiting debug state. 1 = The processor has exited debug state. This is the reset value. The debugger can poll this bit to determine when the processor responds to a request to leave debug state.
[0][a]	Core halted	Core halted bit: 0 = The processor is in normal state. This is the reset value. 1 = The processor is in debug state. The debugger can poll this bit to determine when the processor has entered debug state.

a. These bits always reflect the status of the processor and, therefore they return to their reset values if the particular reset event affects the processor. For example, a core reset event such as **nDBGRESET** sets DSCR[18] to a 0 and DSCR[1:0] to b10.

Internal view

Access is through the Baseline CP14 interface and is read-only

To access the Debug Status and Control Register, read CP14 c1 with:

```
MRC p14, 0, <Rd>, c0, c1, 0 ; Read Debug Status and Control Register
```

External view

Access is through the memory-mapped interface, offset 0x88, and through the Extended CP14 interface.

To access the Debug Status and Control Register, read or write CP14 c1 with:

```
MRC p14, 0, <Rd>, c0, c2, 2 ; Read Debug Status and Control Register
MCR p14, 0, <Rd>, c0, c2, 2 ; Write Debug Status and Control Register
```

DTR access mode

You can use the DTR access mode field to optimize data transfer between a debugger and the processor.

The DTR access mode can be one of the following:

- Nonblocking. This is the default mode
- Stall
- Fast.

In Nonblocking mode, the APB reads from the DTRTX and writes to the DTRRX and ITR are ignored if the appropriate READY flag is not set. In particular:

- writes to DTRRX are ignored if DTRRXfull_l is set
- writes to ITR are ignored if InstrCompl_l is not set
- reads from DTRTX are ignored and return an Unpredictable value if DTRTXfull_l is not set.

The debugger accessing these registers must first read the DSCR, and perform any of the following:

- write to the DTRRX if the DTRRXfull_l flag was cleared
- write to the ITR if the InstrCompl_l flag was set
- read from the DTRTX if the DTRTXfull_l flag was set.

Failure to read the DSCR before one of these operations leads to Unpredictable behavior.

In Stall mode, the APB accesses to DTRRX, DTRTX, and ITR stall under the following conditions:

- writes to DTRRX are stalled until DTRRXfull is cleared
- writes to ITR are stalled until InstrCompl is set
- reads from DTRTX are stalled until DTRTXfull is set.

Fast mode is similar to Stall mode except that in Fast mode, the processor fetches an instruction from the ITR when a DTRRX write or DTRTX read succeeds. In Stall mode and Nonblocking mode, the processor fetches an instruction from the ITR when an ITR write succeeds.

8.5.3 Program Counter Sampling Register (DBGPCSR)

The Cortex-A9 processor implements the DBGPCSR. This register indicates the VA of the latest branch target plus a processor state dependent offset. The bottom two bits encode the processor state so the profiling tool can work out the VA by subtracting the offset.

Figure 8-6 shows the bit arrangements of the DBGPCSR.

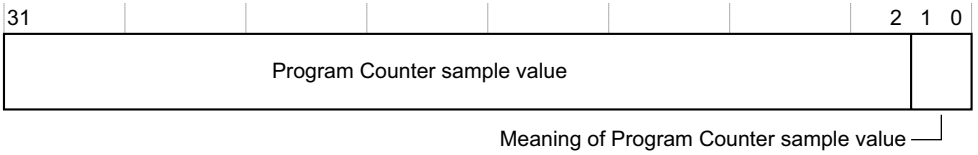


Figure 8-6 Program Counter Sampling Register bit assignments

Table 8-4 shows how the bit values correspond with the DBGPCSR bit functions.

Table 8-4 Program Counter Sampling Register bit assignments

Bits	Name	Description
[31:2]	Program Counter Sample value	The sampled value of bits [31:2] of the PC.
[1:0]	Meaning of PC Sample value	0b00 = References an ARM state instruction. 0bx1 = References a Thumb or ThumbEE state instruction. 0b10 = Implementation defined.

Reads through the Extended CP14 interface of the CP14 register that map to the DBGPCSR are UNPREDICTABLE.

8.5.4 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.

8.5.5 Debug Run Control Register (DBGDRCR)

The DRCR requests the processor to enter or leave debug state. It also clears the sticky exception bits present in the DSCR.

Figure 8-7 shows the bit arrangement of the DRCR.

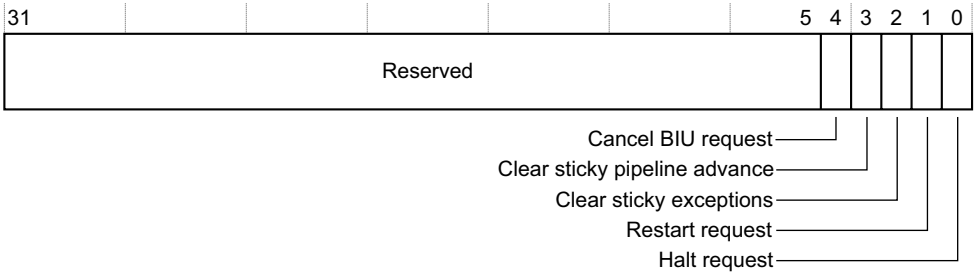


Figure 8-7 Debug Run Control Register bit assignments

Table 8-5 shows how the bit values correspond with the Debug Run Control Register functions.

Table 8-5 Debug Run Control Register bit assignments

Bits	Name	Description
[31:5]	-	RAZ/SBZP.
[4]	Cancel BIU request	Not implemented.
[3]	Clear sticky pipeline advance	Clear sticky pipeline advance. Writing a 1 to this bit clears DSCR[25].

Table 8-5 Debug Run Control Register bit assignments (continued)

Bits	Name	Description
[2]	Clear sticky exceptions	Clear sticky exceptions. Writing a 1 to this bit clears DSCR[8:6].
[1]	Restart request	Restart request. Writing a 1 to this bit requests that the processor leaves debug state. This request is held until the processor exits debug state. When the debugger makes this request, it polls DSCR[1] until it reads 1. This bit always reads as zero. Writes are ignored when the processor is not in debug state.
[0]	Halt request	Halt request. Writing a 1 to this bit triggers a halting debug event, that is, a request that the processor enters debug state. This request is held until the debug state entry occurs. When the debugger makes this request, it polls DSCR[0] until it reads 1. This bit always reads as zero. Writes are ignored when the processor is already in debug state.

8.5.6 Breakpoint Value Registers

The *Breakpoint Value Registers* (BVRs) are registers 64-79, at offsets `0x100-0x13C`. 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 8-6 shows the BVRs and corresponding BCRs.

Table 8-6 BVRs and corresponding BCRs

Breakpoint Value Registers			Breakpoint Control Registers		
Register	**Register number**	**Offset**	**Register**	**Register number**	**Offset**
BVR0	64	`0x100`	BCR0	81	0x140
BVR1	65	`0x104`	BCR1	82	0x144
BVR2	66	`0x108`	BCR2	83	0x148

Table 8-6 BVRs and corresponding BCRs (continued)

Breakpoint Value Registers			Breakpoint Control Registers		
Register	**Register number**	**Offset**	**Register**	**Register number**	**Offset**
BVR3	66	0x10c	BCR3	84	0x14C
BVR4	67	0x110	BCR4	85	0x150
BVR5	68	0x114	BCR5	86	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 8-7 shows how the bit values correspond with the Breakpoint Value Registers functions.

Table 8-7 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. See Chapter 4 *The System Control Coprocessor* on page 4-1 for information on the Context ID Register.

8.5.7 Breakpoint Control Registers

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

- breakpoints
- linked breakpoints.

Figure 8-8 shows the bit arrangement of the BCRs.

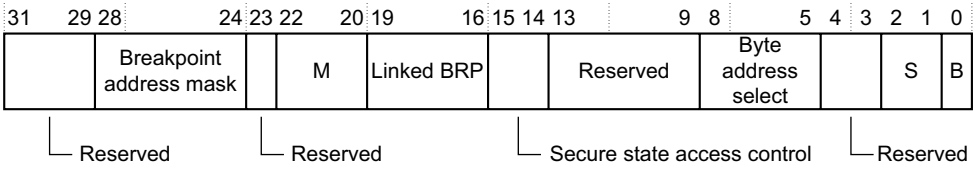


Figure 8-8 Breakpoint Control Registers bit assignments

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

Table 8-8 Breakpoint Control Registers bit functions

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.

Table 8-8 Breakpoint Control Registers bit functions (continued)

Bits	Name	Description
[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.
[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.

Table 8-8 Breakpoint Control Registers bit functions (continued)

Bits	Name	Description
[4:3]	-	RAZ on reads, SBZP on writes.
[2:1]	S	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 8-9 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.

8.5.8 Watchpoint Value Registers

The *Watchpoint Value Registers* (WVRs) are registers 96-111, at offsets 0x180-0x1BC. Each WVR is associated with a *Watchpoint Control Register* (WCR), for example:

- WVR0 with WCR0
- WVR1 with WCR1.

This pattern continues up to WVR5 with WCR5.

Table 8-10 shows the WVRs and corresponding WCRs.

Table 8-10 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
WVR4	100	0x190	WCR4	116	0x1D0
WVR5	101	0x194	WCR5	117	0x1D4

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 8-11 shows how the bit values correspond with the Watchpoint Value Registers functions.

Table 8-11 Watchpoint Value Registers bit functions

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

8.5.9 Watchpoint Control Registers

The WCRs contain the necessary control bits for setting:

- watchpoints
- linked watchpoints.

Figure 8-9 shows the bit arrangement of the Watchpoint Control Registers.

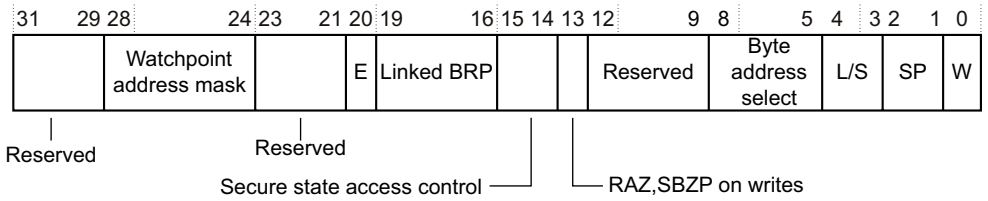


Figure 8-9 Watchpoint Control Registers bit assignments

Table 8-12 shows how the bit values correspond with the Watchpoint Control Registers functions.

Table 8-12 Watchpoint Control Registers bit functions

Bits	Name	Description
[31:29]	-	RAZ on reads, SBZP on writes.
[28:24]	Watchpoint address mask	Watchpoint address mask. RAZ/WI b00000 = no mask
[23:21]	-	RAZ on reads, SBZP on writes.

Table 8-12 Watchpoint Control Registers bit functions (continued)

Bits	Name	Description
[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 8-12 Watchpoint Control Registers bit functions (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]	S	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.

8.5.10 Device Power-down and Reset Control Register (DBGPRCR)

The DBGPRCR Register is a read and write register that controls power-down related functionality. Figure 8-10 shows the format of the DBGPRCR Register.

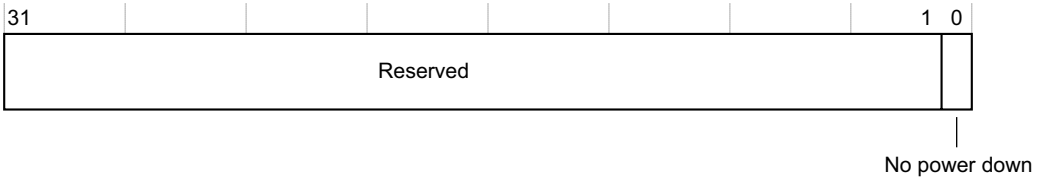


Figure 8-10 DBGPRCR Register bit assignments

Table 8-13 shows DBGPRCR Register bit assignments.

Table 8-13 DBGPRCR bit functions

Bits	Name	Description
[31:1]	Reserved	-
[0]	No power down	When set to 1, the **DBGNOPWRDWN** output signal is HIGH. This output 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. 0 = **DBGNOPWRDWN** is LOW. This is the reset value. 1 = **DBGNOPWRDWN** is HIGH.

8.5.11 Device Power-down and Reset Status Register (DBGPRSR)

The Cortex-A9 processor does not support debug of powered down processors. The DBGPRSR is a read-only register that provides information about the reset and power-down state of the processor. Figure 8-11 shows the bit arrangement of the Device Power-down and Reset Status Register.

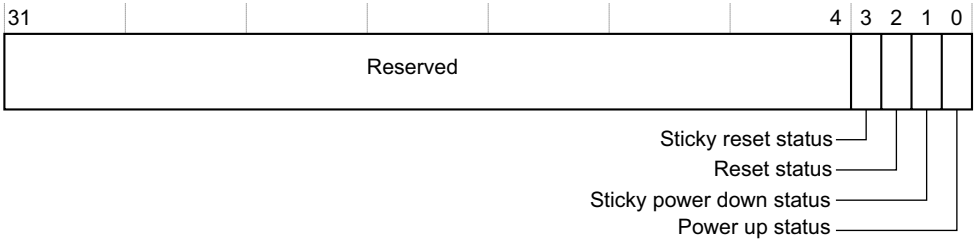


Figure 8-11 Device power-down and reset status register

Table 8-14 shows how the bit values correspond with the device power-down and reset status functions

Table 8-14 Device power-down and reset status register bit assignments

Bits	Name	Description
[31:4]	-	Reserved
[3]	-	Sticky reset status
[2]	-	Reset status
[1]	-	Sticky power-down status. RAZ
[0]	-	Power up status. RAO

8.6 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 8-15 shows the contents of the Management registers for the Cortex-A9 debug unit.

Table 8-15 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	*Integration Mode Control Register (DBGITCTRL)* on page 8-39.
0xF04-0xF9C	961-999	RAZ	-	Reserved for Management Register expansion.
0xFA0	1000	RW	CLAIMSET	*Claim Tag Set Register (DBGCLAIMSET)* on page 8-39.
0xFA4	1001	RW	CLAIMCLR	*Claim Tag Clear Register (DBGCLAIMCLR)* on page 8-40.
0xFA8-0xFBC	1002-1003	-	-	RAZ.
0xFB0	1004	WO	LOCKACCESS	*Lock Access Register (DBGLAR)* on page 8-41.
0xFB4		RO	LOCKSTATUS	*Lock Status Register (DBGLSR)* on page 8-41.
0xFB8		RO	AUTHSTATUS	*Authentication Status Register (DBGAUTHSTATUS)* on page 8-42.
0xFBC-0xFC4	1007-1009	-	-	RAZ.
0xFC8	1010	RO	DEVID	Device Identifier.
0xFCC	1011	RO	DEVTYPE	*Device Type Register (DBGDEVTYPE)* on page 8-43.
0xFD0-0xFFC	1012-1023	R	-	*Identification Registers* on page 8-44.

8.6.1 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 8-16 shows the offset value, register number, mnemonic, and description that are associated with each Process ID Register.

Table 8-16 Processor Identifier Registers

Offset (hex)	Register number	Mnemonic		Register value	Description
0xD00	832	CPUID	RO	Input dependant	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

8.6.2 Integration Mode Control Register (DBGITCTRL)

The read and write Integration Mode Control Register enables the processor to switch from a functional, default mode, into integration mode, where the inputs and outputs of the device can be directly controlled for integration testing or topology detection.

Figure 8-12 shows the bit arrangement of the Integration Mode Control Register.

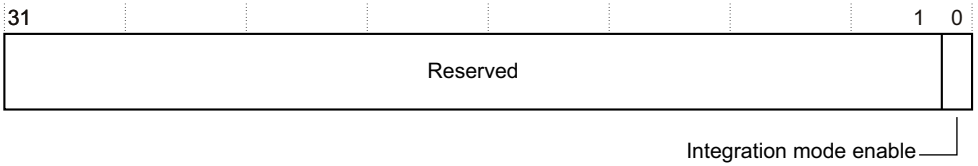


Figure 8-12 Integration Mode Control Register bit assignments

Table 8-17 shows how the bit values correspond with the Integration Mode Control Register functions.

Table 8-17 Integration Mode Control Register bit assignments

Bits	Name	Description
[31:1]	-	RAZ/SBZP.
[0]	Integration mode enable	RAZ/WI Integration mode enable bit: 0 = normal operation, reset value 1 = integration mode enabled. When this bit is set to 1, the processor reverts into integration mode to enable integration testing or topology detection.

8.6.3 Claim Tag Set Register (DBGCLAIMSET)

Bits in the Claim Tag Set Register do not have any specific functionality. The external debugger and debug monitor set these bits to lay claims on debug resources.

Figure 8-13 shows the bit arrangement of the Claim Tag Set Register.

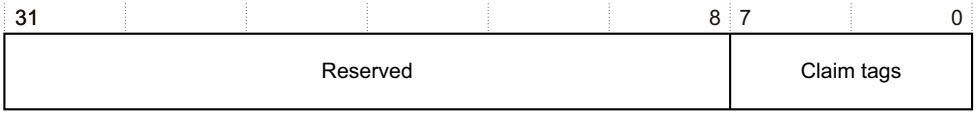


Figure 8-13 Claim Tag Set Register

Table 8-18 shows how the bit values correspond wit the Claim Tag Set Register functions.

Table 8-18 Claim Tag Set Register bit assignments

Bits	Name	Description
[31:8]	-	RAZ on reads, SBZP on writes.
[7:0]	Claim tags	Indicates the claim tags. Writing 1 to a bit in this register sets that particular claim. You can read the claim status at the Claim Tag Clear Register. For example, if you write 1 to bit [3] of this register, bit [3] of the Claim Tag Clear Register is read as 1. Writing 0 to a specific claim tag bit has no effect. This register always reads `0xFF`, indicating that up to eight claims can be set.

8.6.4 Claim Tag Clear Register (DBGCLAIMCLR)

The read and write Claim Tag Clear Register is used to read the claim status on debug resources.

Figure 8-14 shows the bit arrangement of the Claim Tag Clear Register.

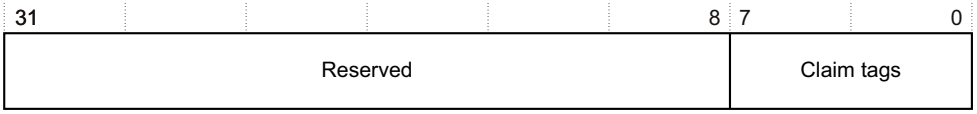


Figure 8-14 Claim Tag Clear Register

Table 8-19 shows how the bit values correspond with the Claim Tag Clear Register functions.

Table 8-19 Claim Tag Clear Register bit assignments

Bits	Name	Description
[31:8]	-	RAZ on reads, SBZP on writes.
[7:0]	Claim tags	Indicates the claim tag status. Writing 1 to a specific claim tag clear bit clears that claim tag. Reading this register returns the current claim tag value. For example, if you write 1 to bit [3] of this register, it is read as 0. The reset value is 0.

8.6.5 Lock Access Register (DBGLAR)

The Lock Access Register is a write-only register that controls writes to the debug registers. The purpose of the Lock Access Register is to reduce the risk of accidental corruption to the contents of the debug registers. It does not prevent all accidental or malicious damage. Because the state of the Lock Access Register is in the debug power domain, it is not lost when the core powers down.

Figure 8-15 shows the bit arrangement of the Lock Access Register.

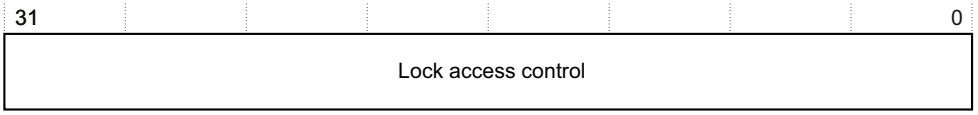


Figure 8-15 Lock Access Register bit assignments

Table 8-20 shows how the bit values correspond with the Lock Access Register functions.

Table 8-20 Lock Access Register bit assignments

Bits	Name	Description
[31:0]	Lock access control	Lock access control. To unlock the debug registers, write a `0xC5ACCE55` key to this register. To lock the debug registers, write any other value. Accesses to locked debug registers are ignored. The reset value is 0.

When this register is written by an external debugger, APB read with **PADDRDBG31**=1, it is write-ignored.

8.6.6 Lock Status Register (DBGLSR)

The Lock Status Register is a read-only register that returns the current lock status of the debug registers.

Figure 8-16 on page 8-42 shows the bit arrangement of the Lock Status Register.

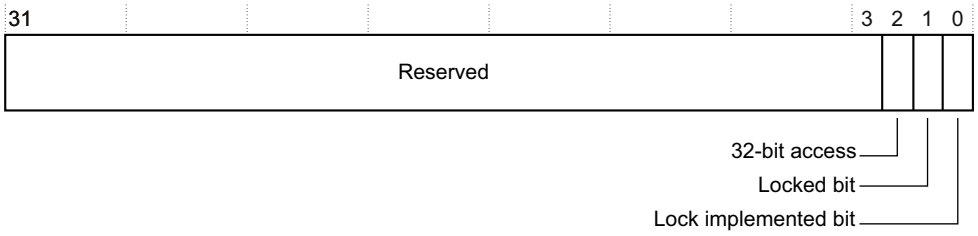


Figure 8-16 Lock Status Register bit assignments

Table 8-21 shows how the bit values correspond with the Lock Status Register functions.

Table 8-21 Lock Status Register bit assignments

Bits	Name	Description
[31:3]	-	Reserved
[2]	32-bit access	Read as zero. It indicates that a 32-bit access is required to write the key to the Lock Access Register.
[1]	Locked bit	This bit indicates the status of the debug registers lock. 0 Lock clear. Debug register writes are permitted. 1 Lock set. Debug register writes are ignored. The Debug reset value of this bit is 1.
[0]	Lock implemented bit	Read-as-One

When this register is read by an external debugger, APB read with **PADDRDBG31**=1, the read value is 0x0.

8.6.7 Authentication Status Register (DBGAUTHSTATUS)

The Authentication Status Register is a read-only register that reads the current values of the configuration inputs that determine the debug permission level.

Figure 8-17 on page 8-43 shows the bit arrangement of the Authentication Status Register.

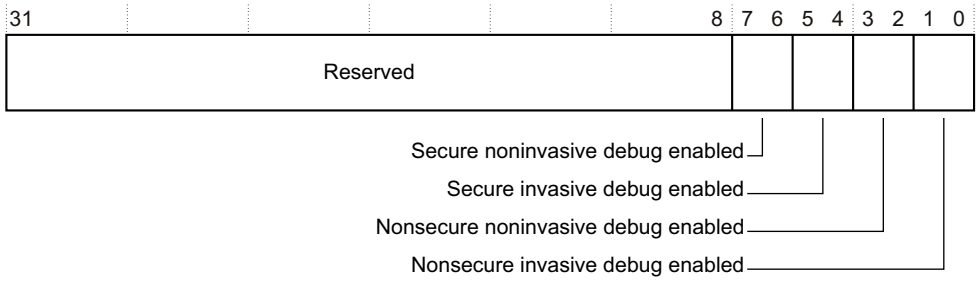


Figure 8-17 Authentication Status Register bit assignments

Table 8-22 shows how the bit values correspond with the Authentication Status Register functions.

Table 8-22 Authentication Status Register bit assignments

Bits	Name	Value	Description
[31:8]	-	-	RAZ
[7]	Secure noninvasive debug enabled	b1	Secure noninvasive debug enable field
[6]		(**DBGEN** || **NIDEN**) && (**SPIDEN** || **SPNIDEN**)	
[5]	Secure invasive debug enabled	b1	Secure invasive debug enable field
[4]		**DBGEN** && **SPIDEN**	
[3]	Non-secure noninvasive debug enabled	b1	Non-secure noninvasive debug enable field
[2]		**DBGEN** || **NIDEN**	
[1]	Non-secure invasive debug enabled	b1	Non-secure invasive debug enable field
[0]		**DBGEN**	

8.6.8 Device Type Register (DBGDEVTYPE)

The Device Type Register is a read-only register that indicates the type of debug component.

Figure 8-18 on page 8-44 shows the bit arrangement of the Device Type Register.

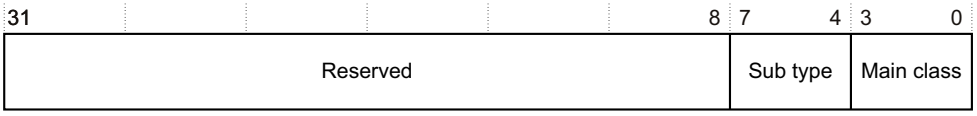


Figure 8-18 Device Type Register bit assignments

Table 8-23 shows how the bit values correspond with the Device Type Register functions.

Table 8-23 Device Type Register bit assignments

Bits	Name	Description
[31:8]	-	RAZ.
[7:4]	Sub type	Indicates that the sub-type of the Cortex-A9 processor is *core*. This value is `0x1`.
[3:0]	Main class	Indicates that the main class of the Cortex-A9 processor is *debug logic*. This value is `0x5`.

8.6.9 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 8-24 shows the offset value, register number, and description that are associated with each Peripheral Identification Register.

Table 8-24 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

Table 8-24 Peripheral Identification Registers (continued)

Offset (hex)	Register number	Description
0xFE4	1017	Peripheral Identification Register 1
0xFE8	1018	Peripheral Identification Register 2
0xFEC	1019	Peripheral Identification Register 3

Table 8-25 shows fields that are in the Peripheral Identification Registers.

Table 8-25 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. The Cortex-A9 processor occupies a single 4KB block, therefore this field is always 0x0.
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. Because the Cortex-A9 processor is designed by ARM Limited, the continuation code is 0x4 and the identity code is 0x3B.
Part number	12 bits	Indicates the part number of the processor. The part number for the Cortex-A9 processor is 0xC08.
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. The current Cortex-A9 revision number is 0x0. The revision number starts at 0x0 and increments by 1 at both major and minor revisions.
RevAnd	4 bits	Indicates the manufacturer revision number. This number starts at 0x0 and increments by the integrated circuit manufacturer on metal fixes. For the Cortex-A9 processor, the initial value is 0x0 but can be changed by the manufacturer.
Customer modified	4 bits	For the Cortex-A9 processor, this value is 0x0.

Table 8-26 shows how the bit values correspond with the Peripheral ID Register 0 functions.

Table 8-26 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. This value is `0x09`.

Table 8-27 shows how the bit values correspond with the Peripheral ID Register 1 functions.

Table 8-27 Peripheral ID Register 1 bit functions

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

Table 8-28 shows how the bit values correspond with the Peripheral ID Register 2 functions.

Table 8-28 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. This value is set to 0.
[3]	This field is always set to `0x1`.
[2:0]	Indicates bits [6:4] of the JEDEC JEP106 Identity Code. This value is set to `0x3`.

Table 8-29 shows how the bit values correspond with the Peripheral ID Register 3 functions.

Table 8-29 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. This value is set to 0.
[3:0]	For the Cortex-A9 processor, this value is set to 0.

Table 8-30 shows how the bit values correspond with the Peripheral ID Register 4 functions.

Table 8-30 Peripheral ID Register 4 bit functions

Bits	Description
[31:8]	RAZ.
[7:4]	Indicates the number of blocks occupied by the Cortex-A9 processor. This field is always set to 0.
[3:0]	Indicates the JEDEC JEP106 Continuation Code. For the Cortex-A9 processor, this value is `0x4`.

Table 8-31 shows the offset value, register number, and value that are associated with each Component Identification Register.

Table 8-31 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

8.7 External debug interface

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

The APB interface is compliant with the AMBA 3 APB interface. This APB slave interface supports 32-bits wide data, stalls, slave-generated aborts, and eleven address bits [12:2] mapping 2x4KB of memory. The **PADDRDBG31** signal indicates to the processor the source of access. See Appendix A *Signal Descriptions* for a complete list of the external debug signals.

Figure 8-19 shows the external debug interface signals.

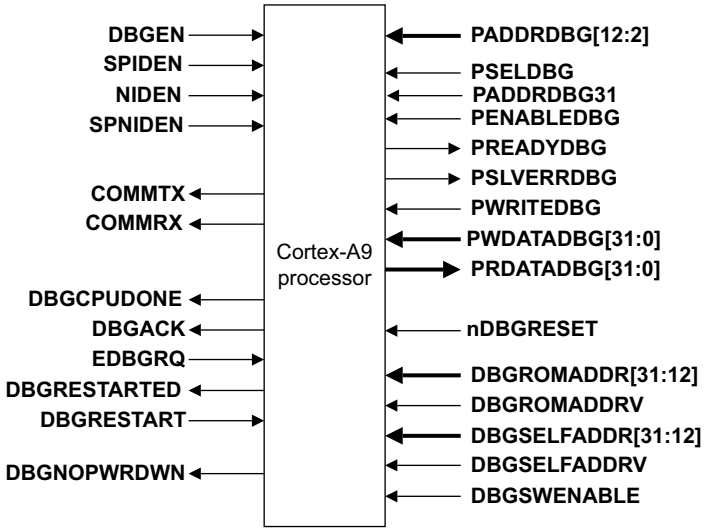


Figure 8-19 External debug interface signals

8.8 Miscellaneous debug signals

This section describes the miscellaneous debug input and output signals in more detail.

8.8.1 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.

8.8.2 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. See *CP14 c1, Debug Status and Control Register (DBGDSCR)* on page 8-16.

8.8.3 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 DSCR[30] DTRRX full flag. See *CP14 c1, Debug Status and Control Register (DBGDSCR)* on page 8-16.

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 DSCR[29] DTRTX full flag. See *CP14 c1, Debug Status and Control Register (DBGDSCR)* on page 8-16.

8.8.4 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.

8.8.5 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.

8.8.6 Authentication signals

Table 8-32 shows a list of the valid combination of authentication signals along with its associated debug permissions. Authentication signals are used to configure the processor so its activity can only be debugged or traced in a certain subset of processor modes and security states.

Table 8-32 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

Table 8-32 Authentication signal restrictions (continued)

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
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.

8.8.7 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. Issue a prefetch flush.

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.

Appendix A
Signal Descriptions

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

A.1 Clock signal

The Cortex-A9 processor has a single externally generated global clock. Table A-1 shows this.

Table A-1 Clock signal for Cortex-A9

Signal	I/O	Description
CLK	I	Global clock

See *Clocking* on page 2-6.

A.2 Resets

Table A-2 shows the reset signals.

Table A-2 Cortex-A9 processor reset signals

Signal	I/O	Description
nCPURESET	I	Cortex-A9 processor reset.
nDBGRESET	I	Cortex-A9 processor debug logic reset.
nDERESET	I	Dedicated data engines reset 0 = not enabled 1 = enabled.

See *Reset* on page 2-9.

A.3 Interrupts

Table A-3 shows the interrupt line signals.

Table A-3 Interrupt line signals

Signal	I/O	Description
nFIQ	I	Cortex-A9 processor FIQ request input lines. 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	Cortex-A9 processor IRQ request input lines. Active-LOW interrupt request: 0 = activate interrupt 1 = do not activate interrupt. The processor treats the **nIRQ** input as level sensitive.

A.4 Configuration

Table A-4 shows the configuration signals.

Table A-4 Configuration signals

Signal	I/O	Description
CFGEND	I	Controls the state of EE bit in the SCTLR: 0 = EE bit is LOW 1 = EE bit is HIGH This pin is only sampled during reset of the processor.
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. This pin is only sampled during reset of the processor.
CP15SDISABLE	I	Disables write access to some system control processor registers: 0 = not enabled 1 = enabled. See *System registers affected by CP15SDISABLE* on page 4-4.
DECLKOFF	I	Controls clocking of the Data Engine during reset sequences 0 = not enabled 1 = enabled.
MAXCLKLATENCY[2:0]	I	Controls dynamic clock gating delays. This pin is sampled during reset of the processor.
TEINIT	I	Default exception handling state: 0 = ARM 1 = Thumb. It sets the SCTLR.TE bit. This pin is only sampled during reset of the processor.
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. This pin is only sampled during reset of the processor.

A.5 Standby and Wait For Event signals

Table A-5 shows standby and wait for event signals.

Table A-5 Standby and wait for event signals

Signal	I/O	Description
EVENTI	I	Event input for Cortex-A9 processor wake-up from WFE state.
EVENTO	O	Event output. This signal is active when one `SEV` instruction is executed.
STANDBYWFI	O	Indicates if the CPU is in WFI mode: 0 = processor not in standby mode 1 = processor in standby mode.
STANDBYWFE	O	Indicates if the CPU is in WFE mode: 0 = processor not in wait for event mode 1 = processor in wait for event mode.

See *Wait for interrupt (WFI/WFE) mode* on page 2-13.

A.6 Power management signals

Table A-6 shows the power management signals.

Table A-6 Power management signals

Signal	I/O	Description
DECLAMP	I	Activates the Cortex-A9 processor clamps: 0 = clamps not active 1 = clamps active.
CPURAMCLAMP	I	Activates the CPU interface clamps: 0 = clamps not active 1 = clamps active.

See *Power management* on page 2-11.

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-7 shows the AXI write address signals for AXI Master0.

Table A-7 AXI-AW signals for AXI Master0

Signal	I/O	Description
AWADDRM0[31:0]	O	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
AWLENM0[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.

Table A-7 AXI-AW signals for AXI Master0 (continued)

Signal	I/O	Description
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] reserved. [7] reserved [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-8 shows the AXI write data signals for AXI Master0.

Table A-8 AXI-W signals for AXI Master0

Signal	I/O	Description
WDATAM0[63:0]	O	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-9 shows the AXI write response signals for AXI Master0.

Table A-9 AXI-B signals for AXI Master0

Signal	I/O	Description
BIDM0[1:0]	I	Response ID
BREADYM0	O	Response ready
BRESPM0[1:0]	I	Write response
BVALIDM0	I	Response valid

Read data channel signals

Table A-10 shows the AXI read address signals for AXI Master0.

Table A-10 AXI-AR signals for AXI Master0

Signal	I/O	Description
ARADDRM0[31:0]	O	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-10 AXI-AR signals for AXI Master0 (continued)

Signal	I/O	Description
ARSIZEM0[1:0]	O	Burst size: b000 = 8-bit transfer b001 = 16-bit transfer b010 = 32-bit transfer b011 = 64-bit transfer.
ARUSERM0[6:0]	O	[6] = Reserved. [5] = Prefetch hint [4:1] Inner attributes 0000 = Strongly-ordered 0001 = Device 0011 = Normal Memory Non-Cacheable 0110 = Write-Through 0111 = Write-Back no Write-Allocate 1111 = Write-Back Write-Allocate [0] shared
ARVALIDM0	O	Address valid.

Read data channel signals

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

Table A-11 AXI-R signals for AXI Master0

Signal	I/O	Description
RVALIDM0	I	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-12 shows the AXI Master0 clock enable signal.

Table A-12 AXI Master0 clock enable signal

Signal	I/O	Source	Description
ACLKENM0	I	Clock generator	Clock enable for the AXI bus that enables the AXI interface to operate at integer ratios of the system clock. See *Clocking* on page 2-6.

A.7.2 AXI Master1 signals

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

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

Read data channel signals

Table A-13 shows the AXI read address signals for AXI Master1.

Table A-13 AXI-AR signals for AXI Master1

Signal	I/O	Description
ARADDRM1[31:0]	O	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.

Table A-13 AXI-AR signals for AXI Master1 (continued)

Signal	I/O	Description
ARLOCKM1[1:0]	O	Lock type: b00 = Normal access.
ARPROTM1[2:0]	O	Protection Type
ARREADYM1	I	Address ready.
ARSIZEM1[1:0]	O	Burst size: b000 = 8-bit transfer b001 = 16-bit transfer b010 = 32-bit transfer b011 = 64-bit transfer.
ARUSERM1[6:0]	O	[6] = Reserved. [5] = Prefetch hint [4:1] = Inner attributes 0000 = Strongly-ordered 0001 = Device 0011 = Normal Memory Non-Cacheable 0110 = Write-Through 0111 = Write-Back no Write-Allocate 1111 = Write-Back Write-Allocate [0] = Shared.
ARVALIDM1	O	Address valid.

Read data channel signals

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

Table A-14 AXI-R signals for AXI Master1

Signal	I/O	Description
RVALIDM1	I	Read valid
RDATAM1[63:0]	I	Read data
RRESPM1[1:0]	I	Read response

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

Signal	I/O	Description
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-15 shows the AXI Master1 clock enable signals.

Table A-15 AXI Master1 clock enable signal

Signal	I/O	Source	Description
ACLKENM1	I	Clock generator	Clock enable for the AXI bus that enables the AXI interface to operate at integer ratios of the system clock. See *Clocking* on page 2-6.

See Chapter 7 *Level 2 Memory Interface*.

A.8 Performance monitoring signals

Table A-16 shows the performance monitoring signals.

Table A-16 Performance monitoring signals

Signal	I/O	Description
DEFLAGS[6:0]	O	Data Engine output flags. Only implemented if the Cortex-A9 processor includes a Data Engine. 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].
PMUEVENT[51:0]	O	Performance Monitoring Unit event bus. See Table A-17.
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-17 gives the correlation between **PMUEVENT** signals and their event numbers.

Table A-17 Event signals and event numbers

Signal	Event number	Description
PMUEVENT[0]	`0x00`	Software increment
PMUEVENT[1]	`0x01`	Instruction cache miss
PMUEVENT[2]	`0x02`	Instruction micro TLB miss

Table A-17 Event signals and event numbers (continued)

Signal	Event number	Description
PMUEVENT[3]	0x03	Data cache miss
PMUEVENT[4]	0x04	Data cache access
PMUEVENT[5]	0x05	Data micro TLB miss
PMUEVENT[6]	0x06	Data read
PMUEVENT[7]	0x07	Data writes
-	0x08	Unused[a]
PMUEVENT[8]	0x68	b00 no instructions renamed b01 one instruction renamed
PMUEVENT[9]	0x68	b00 no instructions renamed b10 two instructions renamed.
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/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]

Table A-17 Event signals and event numbers (continued)

Signal	Event number	Description
PMUEVENT[23]	0x51	Coherent linefill hit[c]
PMUEVENT[24]	0x60	Instruction cache dependent stall
PMUEVENT[25]	0x61	Data cache dependent stall
PMUEVENT[26]	0x62	Main TLB miss stall
PMUEVENT[27]	0x63	STREX passed
PMUEVENT[28]	0x64	STREX failed
PMUEVENT[29]	0x65	Data eviction
PMUEVENT[30]	0x66	Issue does not dispatch any instruction
PMUEVENT[31]	0x67	Issue is empty
PMUEVENT[32]	0x70	Main Execution Unit pipe
PMUEVENT[33]	0x71	Second Execution Unit pipe
PMUEVENT[34]	0x72	Load/Store pipe
PMUEVENT[35]	0x73	b00 no floating-point instruction renamed b01 one floating-point instruction renamed
PMUEVENT[36]	0x73	b00 no floating-point instruction renamed b10 two floating-point instructions renamed
PMUEVENT[37]	0x74	b00 no NEON instruction renamed b01 one NEON instruction renamed
PMUEVENT[38]	0x74	b00 no NEON instruction renamed b10 two NEON instructions renamed
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

Table A-17 Event signals and event numbers (continued)

Signal	Event number	Description
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

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:

- *Predefined events summary* on page 4-102
- *Jazelle events* on page 4-104
- *Cortex-A9 specific events* on page 4-105.

A.9 Parity signal

Table A-18 shows the parity signal. See *Parity error support* on page 6-13.

Table A-18 Parity signal

Signal	I/O	Description
PARITYFAIL[7:0]	O	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.10 MBIST interface

Table A-19 shows the MBIST interface signals.

Table A-19 MBIST interface signals

Signal	I/O	Description
MBISTADDR[10:0]	I	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-20 shows these signals with parity support implemented.

Table A-20 MBIST signals with parity support implemented

Signal	I/O	Description
MBISTBE[31:0]	I	MBIST write enable
MBISTINDATA[71:0]	I	MBIST data in
MBISTOUTDATA[287:0]	O	MBIST data out

Table A-21 shows these signals without parity support implemented.

Table A-21 MBIST signals without parity support implemented

Signal	I/O	Description
MBISTBE[25:0]	I	MBIST write enable
MBISTINDATA[63:0]	I	MBIST data in
MBISTOUTDATA[255:0]	O	MBIST data out

See the *Cortex-A9 r0p0 MBIST TRM* for a description of MBIST.

A.11 Scan test signal

Table A-22 lists the scan test signal.

Table A-22 Scan test signal

Signal	I/O	Description
SE	I	Scan enable: 0 = not enabled 1 = enabled.

A.12 External Debug interface

The following sections describe the external debug interface signals:

- *Authentication interface*
- *APB interface signals* on page A-24
- *CTI signals* on page A-25
- *Miscellaneous debug interface signals* on page A-26.

A.12.1 Authentication interface

Table A-23 shows the authentication interface signals.

Table A-23 Authentication interface signals

Signal	I/O	Description
DBGEN	I	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.12.2 APB interface signals

Table A-24 shows the APB interface signals.

Table A-24 APB interface signals

Signal	I/O	Description
PENABLEDBG	I	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.12.3 CTI signals

Table A-25 shows the CTI signals.

Table A-25 CTI signals

Signal	I/O	Description
EDBGRQ	I	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.12.4 Miscellaneous debug interface signals

Table A-26 shows the miscellaneous debug interface signals.

Table A-26 Miscellaneous debug signals

Signal	I/O	Description
COMMRX	O	Communications channel receive. Receive portion of Data Transfer Register full flag: 0 = empty 1 = full.
COMMTX	O	Communications channel transmit. Transmit portion of Data Transfer Register full flag: 0 = empty 1 = full.
DBGNOPWRDWN	O	Debugger has requested the Cortex-A9 processor is not powered down.
DBGSWENABLE	I	When HIGH only the external debug agent can modify debug registers. 0 = not enabled. This is the default. 1 = enabled.
DBGROMADDR[31:12]	I	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[31:12]	I	Specifies bits [31:12] 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 8 *Debug*.

A.13 PTM interface signals

Table A-27 shows the PTM interface signals.

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-27 PTM interface signals

Signal	I/O	Description
WPTCOMMIT[1:0]	O	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 b1101 = 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.
WPTPC[31:0]	O	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.

Table A-27 PTM interface signals (continued)

Signal	I/O	Description
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. See *Processor operating states* on page 3-5 for information about the J bit.
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. See *Processor operating states* on page 3-5 for information about the T bit and the J bit.
WPTTARGETTBIT	O	T bit for waypoint destination See *Processor operating states* on page 3-5 for information about the T bit.
WPTTRACEPROHIBITED	O	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. 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.

Table A-27 PTM interface signals (continued)

Signal	I/O	Description
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.
WPTVALID	O	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 2-4.

Appendix B
AC Characteristics

This chapter gives the timing diagram and timing parameters for the Cortex-A9 processor. It contains the following sections:

- *Cortex-A9 timing* on page B-2
- *Cortex-A9 signal timing parameters* on page B-3.

B.1 Cortex-A9 timing

The AMBA bus interface of the Cortex-A9 processor conforms to the *AMBA Specification*. See the *AMBA Specification* for the relevant timing diagrams for the Cortex-A9 processor.

B.2 Cortex-A9 signal timing parameters

Signal timing parameters are given in:

- *Registered signals*
- *Unregistered signals*.

B.2.1 Registered signals

To ensure ease of integration of the Cortex-A9 processor into embedded applications, and to simplify synthesis flow, the following design techniques have been used:

- a single rising edge clock times all activity
- all signals and buses are unidirectional
- all inputs are required to be synchronous to the relevant clock, CLK, or HCLK.

These techniques simplify the definition of the Cortex-A9 processor top-level signals because all outputs change from the rising edge and all inputs are sampled with the rising edge of the clock. In addition, all signals are either input or output only. Bidirectional signals are not used.

B.2.2 Unregistered signals

The unregistered input signals are:

- **ARREADYM0**, **ARREADYM1**
- **RVALID0**, **RVALID1**
- **AWREADY0**, **AWREADY1**
- **WREADY0**, **WREADY1**
- **BVALID0**, **BVALID1**
- **nIRQ**
- **CLK**, **ACLKEN**, **nCPURESET**, **nDERESET**, and **nDBGRESET**
- **CPUCLKOFF** and **DECLKOFF.**

There are no unregistered output signals.

Figure B-1 on page B-4 shows the Cortex-A9 timing parameters for unregistered signals. The timing parameter T is the internal clock latency of the clock buffer tree, and it is dependent on process technology and design parameters. Timing parameters ending with suffix h represent hold times. Timing parameters ending with suffix d represent delay times. Contact your silicon supplier for more details.

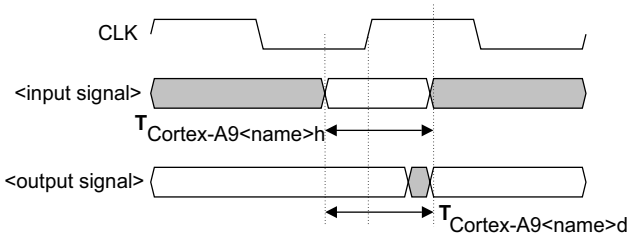


Figure B-1 Cortex-A9 timing parameters for unregistered signals

Note

Actual clock frequencies and input and output timing constraints vary according to application requirements and the silicon process technologies used. The maximum operating clock frequencies attained by ARM devices increases over time as a result.

Appendix C
Instruction Cycle Timings

This chapter describes the cycle timings of integer instructions on Cortex-A9 processors. It contains the following sections:

C.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.

C.2 Data-processing instructions

Table C-1shows the execution unit cycle time for data-processing instructions.

Table C-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 C-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 also *Serializing instructions* on page C-10	1	-	-

C.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 C-2 shows cycle timing for single load and store operations. The result latency is the latency of the first loaded register.

Table C-2 Single load and store operation cycle timings

Instruction cycles	AGU cycles	Result latency	
		Fast forward cases	other cases
LDR ,[reg] LDR ,[reg imm] LDR ,[reg reg] LDR ,[reg reg LSL #2]	1	2	3
LDR ,[reg reg LSL reg] LDR ,[reg reg LSR reg] LDR ,[reg reg ASR reg] LDR ,[reg reg ROR reg] LDR ,[reg reg, RRX]	1	3	4
LDRB ,[reg] LDRB ,[reg imm] LDRB ,[reg reg] LDRB ,[reg reg LSL #2] LDRH ,[reg] LDRH ,[reg imm] LDRH ,[reg reg] LDRH ,[reg reg LSL #2]	2	3	4
LDRB ,[reg reg LSL reg] LDRB ,[reg reg ASR reg] LDRB ,[reg reg LSL reg] LDRB ,[reg reg ASR reg] LDRH ,[reg reg LSL reg] LDRH ,[reg reg ASR reg] LDRH ,[reg reg LSL reg] LDRH ,[reg reg ASR reg]	2	4	5

The Cortex-A9 processor can load or store two 32-bit registers in each cycle. However, to access 64 bits, the address must be 64-bit aligned.

This scheduling is done in the *Address Generation Unit* (AGU). The number of cycles required by the AGU to process the load multiple or store multiple operations depends on the length of the register list and the 64-bit alignment of the address. The resulting latency is the latency of the first loaded register. Table C-3 shows the cycle timings for load multiple operations.

Table C-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 C-4 shows the cycle timings of store multiple operations.

Table C-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

C.4 Multiplication instructions

Table C-4 on page C-7 shows the cycle timings for multiplication instructions.

Table C-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

C.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 C-3.
- Load instructions to the PC register are processed in the execution units as standard instructions. See *Load and store instructions* on page C-4.

Also, see *About the L1 instruction side memory system* on page 6-6 for some information on dynamic branch prediction.

C.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.

C.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 D
Revisions

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

Table D-1 Issue A

Change	Location
First release	-

Table D-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 6-6 • *Branch instructions* on page C-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 4-98.
MIDR bits[3:0] updated from 0 to 1	Table 4-27 on page 4-33.
ID_MMFR3 [23:20] bit value corrected to 0x1	Table 4-38 on page 4-48.
AFE bit description corrected	Table 4-47 on page 4-60.
Auxiliary Control Register bit field corrections	• Table 4-48 on page 4-64 • Figure 4-34 on page 4-64.
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 D-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 6-13.
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 8-1 on page 8-10.
DBGDSCR description corrected with the addition of internal and external view descriptions.	*CP14 c1, Debug Status and Control Register (DBGDSCR)* on page 8-16.
MOE bits descriptions re-ordered and extended	Table 8-3 on page 8-18.
Additional cross-references added from Table 10-1	• *Debug State Cache Control Register (DBGDSCCR)* on page 8-25 • *CP14 c1, Debug Status and Control Register (DBGDSCR)* on page 8-16 • *Device Power-down and Reset Status Register (DBGPRSR)* on page 8-35 • *Integration Mode Control Register (DBGITCTRL)* on page 8-39 • *Claim Tag Clear Register (DBGCLAIMCLR)* on page 8-40 • *Lock Access Register (DBGLAR)* on page 8-41 • *Lock Status Register (DBGLSR)* on page 8-41 • *Authentication Status Register (DBGAUTHSTATUS)* on page 8-42 • *Device Type Register (DBGDEVTYPE)* on page 8-43.
Table 10-1 footnotes corrected	Table 8-1 on page 8-10.
Byte address field entries corrected.	Table 8-12 on page 8-32.
Interrupts signals descriptions corrected	Table A-3 on page A-4.
AXI USER descriptions extended	• Table A-7 on page A-8 • Table A-10 on page A-11 • Table A-13 on page A-13.

Table D-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.	-
The power management signal **BISTSCLAMP** is removed.	-
Dynamic high level clock gating added.	*Dynamic high level clock gating* on page 2-7
TLB information updated.	Table 1-1 on page 1-8, Table 4-15 on page 4-23, Table 4-29 on page 4-36
ID_MMF3[15:12] description shortened.	*Memory Model Features Register 3* on page 4-47
ACTLR updated to include reference to PL310 optimizations.	*Auxiliary Control Register* on page 4-63
Addition of a second replacement strategy. Selection done by SCTLR.RR bit.	*System Control Register* on page 4-59
Event information extended.	*Event Selection Register* on page 4-101
DEFLAGS[6:0] added.	*DEFLAGS[6:0]* on page 4-108, *Performance monitoring signals* on page A-16
Power Control Register description added.	*Power Control Register* on page 4-125
PL310 optimizations added to L2 memory interface description	*Optimized accesses to the L2 memory interface* on page 7-7
Added debug request restart diagram.	*Effects of resets on debug registers* on page 8-7
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* on page A-5

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

Change	Location
PMUEVENT bus description extended.	*Performance monitoring signals* on page A-16
PMUSECURE and **PMUPRIV** added.	*Performance monitoring signals* on page A-16
Description of serializing behavior of `DMB` updated.	*Serializing instructions* on page C-10

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.

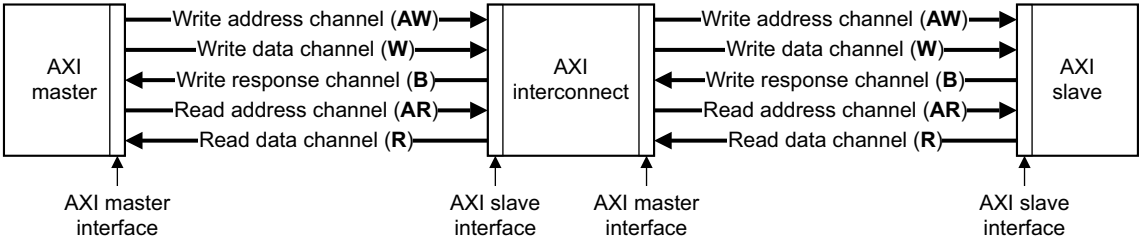
Automatic Test Pattern Generation (ATPG)
The process of automatically generating manufacturing test vectors for an ASIC design, using a specialized software tool.

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.