Cortex-A5™

Revision: r0p1

Technical Reference Manual

ARM®

ARM DDI 0433C (ID040516)

Cortex-A5
Technical Reference Manual

Release Information

The following changes have been made to this book.

Change history

Date	Issue	Confidentiality	Change
24 December 2009	A	Non-Confidential	First release for r0p0
30 September 2010	B	Non-Confidential	First release for r0p1
04 January 2016	C	Non-Confidential	Second release for r0p1

Proprietary Notice

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

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

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

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

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

Confidentiality Status

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

Unrestricted Access is an ARM internal classification.

Product Status

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

Web Address

`http://www.arm.com`

Contents
Cortex-A5 Technical Reference Manual

List of Tables
Cortex-A5 Technical Reference Manual

List of Figures
Cortex-A5 Technical Reference Manual

Preface

This preface introduces the *Cortex-A5 Technical Reference Manual*. It contains the following sections:

- *About this book* on page xii
- *Feedback* on page xvi.

About this book

This book is for the Cortex-A5 processor.

Product revision status

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

r*n* Identifies the major revision of the product.

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

Intended audience

This book is written for hardware and software engineers implementing Cortex-A5 system designs. It provides information that enables designers to integrate the processor into a target system.

Using this book

This book is organized into the following chapters:

Chapter 1 *Introduction*

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

Chapter 2 *Functional Description*

Read this for a description of the functionality of the processor.

Chapter 3 *Programmers Model*

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

Chapter 4 *System Control*

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

Chapter 5 *Non-debug Use of CP14*

Read this for a description of the CP14 coprocessor and its non-debug uses.

Chapter 6 *Memory Management Unit*

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

Chapter 7 *Level 1 Memory System*

Read this for a description of the *Level 1* (L1) memory system, including caches, *Translation Lookaside Buffers* (TLB), and store buffer.

Chapter 8 *Level 2 Memory Interface*

Read this for a description of the *Level 2* (L2) memory interface and the AXI interface attributes.

Chapter 9 *Debug*

Read this for a description of the Cortex-A5 support for debug.

Chapter 10 *Performance Monitoring Unit*

Read this for a description of the Cortex-A5 *Performance Monitoring Unit* (PMU) and associated events.

Appendix A *Signal Descriptions*

Read this for a description of the signals.

Appendix B *Revisions*

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

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

Conventions

Conventions that this book can use are described in:

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

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.

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

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

`monospace bold` Denotes language keywords when used outside example code.

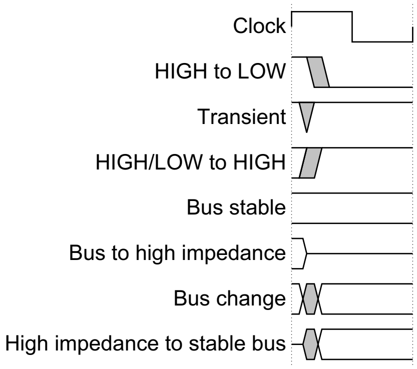
< and > Enclose replaceable terms for assembler syntax where they appear in code or code fragments. For example:

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

Timing diagrams

The figure named *Key to timing diagram conventions* on page xiv 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.

Additional reading

This section lists publications by ARM and by third parties.

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

ARM publications

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

- *Cortex-A5 Supplementary Datasheet* (ARM DDI 0448)
- *Cortex-A5 Floating-Point Unit (FPU) Technical Reference Manual* (ARM DDI 0449)
- *Cortex-A5 NEON Media Processing Engine Technical Reference Manual* (ARM DDI 0450)
- *Cortex-A5 Configuration and Sign-Off Guide (*ARM DII 0210)
- *Cortex-A5 Integration Manual* (ARM DIT 0001)
- *Cortex-A5 MPCore Technical Reference Manual* (ARM DDI 0434)
- *ARM Architecture Reference Manual, ARMv7-A and ARMv7-R edition* (ARM DDI 0406)
- *ARM Architecture Reference Manual Performance Monitors v2 Supplement* (ARM DDI 0457)
- *CoreSight ETM™-A5 Technical Reference Manual* (ARM DDI 0435)
- *CoreSight ETM™-A5 Configuration and Sign-Off Guide* (ARM DII 0212)
- *CoreSight ETM™-A5 Integration Manual* (ARM DIT 0002)

- *CoreSight Design Kit for Cortex-A5 Integration Manual* (ARM DIT 0003)
- *CoreSight Embedded Trace Macrocell v3.5 Architecture Specification* (ARM IHI 0014)
- *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)
- *Cortex-A5 Release Note* (AT550-DC-06003)
- *Cortex-A5 Floating-Point Unit (FPU) Release Note* (AT551-DC-06001)
- *Cortex-A5 NEON Media Processing Engine Release Note* (AT552-DC-06001).

Other publications

This section lists relevant documents published by third parties:

- *ANSI/IEEE Std 754-2008, IEEE Standard for Binary Floating-Point Arithmetic*
- *IEEE Std. 1500-2005, IEEE Standard Testability Method for Embedded Core-based Integrated Circuits.*

Feedback

ARM welcomes feedback on this product and its documentation.

Feedback on this product

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

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

Feedback on content

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

- the title
- the number, ARM DDI 0433C
- the page numbers to which your comments apply
- a concise explanation of your comments.

ARM also welcomes general suggestions for additions and improvements.

Chapter 1
Introduction

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

1.1 About the Cortex-A5 processor

The Cortex-A5 processor is a high-performance, low-power, ARM macrocell with an L1 cache subsystem that provides full virtual memory capabilities. The Cortex-A5 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.

1.1.1 Floating-Point Unit

The *Floating-Point Unit* (FPU) supports the ARMv7 VFPv4-D16 architecture without Advanced SIMD extensions (NEON). It is tightly integrated to the Cortex-A5 processor pipeline. It provides trapless execution and is optimized for scalar operation. It can generate an Undefined instruction exception on vector instructions that enables the programmer to emulate vector capability in software.

The design can include the FPU only, in which case the *Media Processing Engine* (MPE) is not included.

See the *Cortex-A5 Floating-Point Unit Technical Reference Manual*.

1.1.2 Media Processing Engine

The 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 included.

See the *Cortex-A5 NEON Media Processing Engine Technical Reference Manual*.

1.1.3 System design components

This section describes the peripheral components used in Cortex-A5 designs:

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

PrimeCell Level 2 Cache Controller (PL310)

The addition of an on-chip secondary cache, also referred to as a Level 2 or L2 cache, is a recognized method of improving the performance of ARM-based systems when significant memory traffic is generated by the processor. The PrimeCell Level 2 Cache Controller reduces the number of external memory accesses and has been optimized for use with the Cortex-A5 processor.

1.2 Variants

The Cortex-A5 processor is available as a uniprocessor, as described in this manual, or an MPCore multiprocessor, as described in the *Cortex-A5 MPCore Technical Reference Manual*.

1.3 Compliance

The Cortex-A5 processor implements the ARMv7-A architecture and includes the following features:

- Thumb®-2 technology 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 3-10. See the *ARM Architecture 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-A5 NEON Media Processing Engine Technical Reference Manual* for implementation-specific information.
- *Vector Floating-Point v4* (VFPv4) architecture for floating-point computation that is compliant with the IEEE 754 standard. See the *ARM Architecture Reference Manua*l for details of the VFPv4 subarchitecture.

  See the *Cortex-A5 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-A5 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 9 *Debug* for more information.

1.4 Features

The Cortex-A5 processor features are:

- in-order pipeline with dynamic branch prediction
- ARM, Thumb, and ThumbEE instruction set support
- TrustZone security extensions
- Harvard level 1 memory system with a *Memory Management Unit* (MMU)
- 64-bit AXI master interface
- ARM v7 debug architecture
- trace support through an *Embedded Trace Macrocell* (ETM) interface
- *Intelligent Energy Manager* (IEM) support with
  - — asynchronous AXI wrappers
  - — two voltage domains
- optional VFPv4-D16 FPU with trapless execution
- optional *Media Processing Engine* (MPE) with NEON technology
- optional Jazelle hardware acceleration.

1.5 Interfaces

The Cortex-A5 processor has the following external interfaces:

- AMBA AXI interface
- APB CoreSight interface
- ETM interface
- *Design for Test* (DFT) interface.

See the following for more information on these interfaces:

- *AMBA AXI Protocol Specification*
- *CoreSight Architecture Specification*
- *CoreSight Embedded Trace Macrocell v3.5 Architecture Specification*.

1.6 Configurable options

Table 1-1 shows the Cortex-A5 processor RTL configurable options.

Table 1-1 Configurable options for the Cortex-A5 processor

Feature	Range of options	Default value
Instruction cache size	4KB, 8KB, 16KB, 32KB, or 64KB	Implementation dependent
Data cache size	4KB, 8KB, 16KB, 32KB, or 64KB	Implementation dependent
Floating Point Unit or Media Processing Engine (NEON)	None, VFPv4-D16, or VFPv4 and NEON	Not included
Jazelle Architecture Extension	Full or trivial	Trivial

1.7 Test features

The Cortex-A5 processor is delivered as fully-synthesizable RTL and is a fully-static design. Scan-chains and test wrappers for production test can be inserted into the design by the synthesis tools during implementation. See the relevant reference methodology documentation for more information.

Production test of the processor cache and TLB RAMs can be performed through a dedicated MBIST interface. See the *Cortex-A5 Integration Manual* for more information about this interface, and how to control it.

1.8 Product documentation, design flow, and architecture

This section describes the Cortex-A5 books, how they relate to the design flow, and the relevant architectural standards and protocols.

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

1.8.1 Documentation

The Cortex-A5 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-A5 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-A5 processor is implemented and integrated.

If you are programming the Cortex-A5 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.

Integration Manual

The *Integration Manual* (IM) describes how to integrate the Cortex-A5 processor into a SoC. It includes a description of the pins that the integrator must tie off to configure the macrocell for the required integration. Some of the integration is affected by the configuration options used when implementing the Cortex-A5 processor.

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

1.8.2 Design flow

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

1. Implementation. The implementer configures and synthesizes the RTL to produce a hard macrocell. If appropriate, this includes integrating the RAMs into the design.
2. Integration. The integrator connects the implemented design into a SoC. This includes connecting it to a memory system and peripherals.

3. Programming. The system programmer develops the software required to configure and initialize the processor, and tests the required application software.

Each stage of the process:

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

  Build configuration

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

  Configuration inputs

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

  Software configuration

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

Note

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

1.8.3 Architecture and protocol information

The Cortex-A5 processor complies with, or implements, the specifications described in:

- *ARM architecture*
- *Trace macrocell*
- *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-A5 processor implements the ARMv7-A architecture profile. See *ARM Architecture Reference Manual, ARMv7-A and ARMv7-R edition*.

Trace macrocell

The Cortex-A5 processor implements ETM architecture version 3.5. See *CoreSight Embedded Trace Macrocell v3.5 Architecture Specification*.

Advanced Microcontroller Bus Architecture

The Cortex-A5 processor complies with the AMBA 3 protocol. See *AMBA AXI Protocol v1.0 Specification* and *AMBA 3 APB Protocol Specification*.

1.9 Product revisions

This section describes the differences in functionality between product revisions:

r0p0 First release.

r0p0-r0p1 No functional changes.

Chapter 2
Functional Description

This chapter describes the functionality of the Cortex-A5 processor. It contains the following sections:

2.1 About the functions

The Cortex-A5 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 functional diagram of the Cortex-A5 processor.

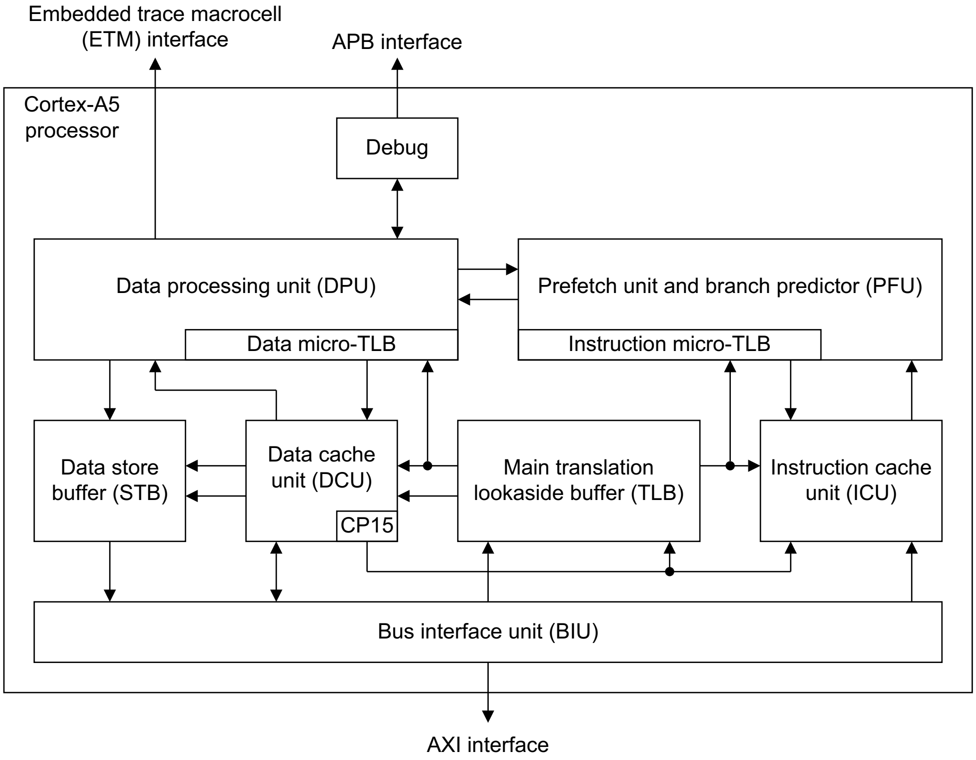


Figure 2-1 Cortex-A5 processor top-level diagram

The following sections describe the blocks and their functions:

- *Data Processing Unit*
- *System control coprocessor* on page 2-3
- *Instruction side memory system* on page 2-3
- *Data side memory system* on page 2-3
- *L1 memory system* on page 2-5
- *L2 AXI interfaces* on page 2-5
- *Media Processing Engine* on page 2-6
- *Floating-Point Unit* on page 2-6
- *Debug* on page 2-6
- *Performance monitoring* on page 2-6
- *Virtualization extensions* on page 2-6.

2.1.1 Data Processing Unit

The *Data Processing Unit* (DPU) holds most of the program-visible state of the processor, such as general-purpose registers, status registers and control registers. It decodes and executes instructions, operating on data held in the registers in accordance with the ARM Architecture. Instructions are fed to the DPU from the *PreFetch Unit* (PFU). The DPU performs instructions that require data to be transferred to or from the memory system by interfacing to the *Data Cache Unit* (DCU), which manages all load and store operations. See Chapter 3 *Programmers Model* for more information.

2.1.2 System control coprocessor

The system control coprocessor, CP15, provides configuration and control of the memory system and its associated functionality.

See Chapter 4 *System Control* for more information.

2.1.3 Instruction side memory system

The instruction side memory system is described in:

- *Instruction Cache Unit*
- *Prefetch Unit*.

Instruction Cache Unit

The *Instruction Cache Unit* (ICU) contains the Instruction Cache controller and its associated linefill buffer. The Cortex-A5 ICache is two-way set associative and uses *Virtually Indexed Physically Tagged* (VIPT) cache-lines holding up to eight ARM or up to sixteen Thumb instructions.

Prefetch Unit

The *Prefetch Unit* (PFU) obtains instructions from the instruction cache or from external memory and predicts the outcome of branches in the instruction stream, then passes the instructions to the DPU for processing. In any given cycle, up to a maximum of four instructions can be fetched and two can be passed to the DPU.

Branch Target Address Cache

The PFU also contains a small four-entry deep *Branch Target Address Cache* (BTAC) used to predict the target address of certain indirect branches. The BTAC implementation is architecturally transparent, so it does not have to be flushed on a context switch.

Branch prediction

The branch predictor is a global type that uses history registers and a 256-entry pattern history table.

Return stack

The PFU includes a 4-entry return stack to accelerate returns from procedure calls. For each procedure call, the return address is pushed onto a hardware stack. When a procedure return is recognized, the address held in the return stack is popped, and the PFU uses it as the predicted return address. The return stack is architecturally transparent, so it does not have to be flushed on a context switch

2.1.4 Data side memory system

The data side memory system is described in:

- *Data Cache Unit* on page 2-4
- *Store Buffer* on page 2-4
- *Bus Interface Unit and AXI interface* on page 2-5.

Data Cache Unit

The *Data Cache Unit* (DCU) consists of the following sub-blocks:

- The *Level 1* (L1) data cache controller, which generates the control signals for the associated embedded tag, data, and dirty memory (RAMs) and arbitrates between the different sources requesting access to the memory resources. The data cache is 4-way set associative and uses a *Physically Indexed Physically Tagged* (PIPT) scheme for lookup which enables unambiguous address management in the system.
- The load/store pipeline which interfaces with the DPU and main TLB.
- The system coprocessor (CP15) controller which performs cache maintenance operations directly on the data cache and indirectly on the instruction cache through an interface with the ICU.

Data cache operation is described in:

- *Read allocate mode*
- *Data cache invalidate on reset*.

Read allocate mode

The Cortex-A5 data cache only supports a write-back policy. It normally allocates a cache line on either a read miss or a write miss. However, there are some situations where allocating on writes is undesirable, such as executing the C standard library memset() function to clear a large block of memory to a known value. Writing large blocks of data like this can pollute the cache with unnecessary data. It can also waste power and performance if a linefill must be performed only to discard the linefill data because the entire line was subsequently written by the memset().

To prevent this, the Cortex-A5 *Bus Interface Unit* (BIU) includes logic to detect when a full cache line has been written by the core before the linefill has completed. If this situation is detected on three consecutive linefills, it switches into read allocate mode. When in read allocate mode, loads behave as normal and can still cause linefills, and writes still lookup in the cache but, if they miss, they write out to L2 rather than starting a linefill.

The BIU continues in read allocate mode until it detects either a cacheable write burst to L2 that is not a full cache line, or there is a load to the same line as is currently being written to L2.

Data cache invalidate on reset

The ARMv7 *Virtual Memory System Architecture* (VMSA) does not support a CP15 operation to invalidate the entire data cache. If this function is required in software, it must be constructed by iterating over the cache geometry and executing a series of individual CP15 invalidate by set/way instructions.

In normal usage the only time the entire data cache has to be invalidated is on reset. The Cortex-A5 processor provides this functionality by default. If it is not required on reset, for example when leaving dormant mode, the invalidate operation can be disabled by asserting and holding the external **L1RSTDISABLE** signal when the reset signal is deasserted.

In parallel to the data cache invalidate, the DCU also sends an invalidate-all request to the ICU and the TLB, unless **L1RSTDISABLE** is asserted.

Store Buffer

The *Store Buffer* (STB) holds store operations when they have left the load/store pipeline and have been committed by the DPU. From the STB, a store can request access to the cache RAMs in the DCU, request the BIU to initiate linefills, or request the BIU to write the data out on the external AMBA AXI interface.

The STB can merge several store transactions into a single transaction if they are to the same 64-bit aligned address.

Bus Interface Unit and AXI interface

The *Bus Interface Unit* (BIU) contains the external master interfaces and various buffers to decouple the interface from the cache and STB. The BIU provides a single 64-bit AMBA AXI master port, which is shared between the instruction side and the data side.

The Cortex-A5 processor supports synchronous clocking at integer ratios of the internal core frequency.

For more information on the AXI IDs, see *AXI transaction IDs* on page 8-3.

The **ARCACHE** and **AWCACHE** signals on the AXI address channels provide the memory type and outer cacheability attributes. It is sometimes necessary to connect an L2 cache that uses the inner cacheability attributes. In the Cortex-A5 processor, this information is provided on the **ARUSER** and **AWUSER** signals, as described in *AXI user bits* on page 8-4.

The **AWUSER** signal also includes additional bits to support an L2 cache in exclusive mode, as described in *AXI user bits* on page 8-4. These encodings are suitable for connecting to the PrimeCell Level 2 Cache Controller. See the *PrimeCell Level 2 Cache Controller (PL310) Technical Reference Manual*.

There is a write buffer that is used to hold data from cache evictions or non-cacheable write bursts before they are written out on the AXI interface.

2.1.5 L1 memory system

The processor L1 memory system includes the following features:

- separate instruction and data caches
- export of memory attributes for L2 memory system.

The caches have the following features:

- Support for independent configuration of the instruction and data cache sizes between 4KB and 64KB.
- Pseudo-random cache replacement policy.
- Ability to disable each cache independently.
- Streaming of sequential data from `LDM` and `LDRD` operations, and sequential instruction fetches.
- Critical word first filling of the cache on a cache miss.
- Implementation of all the cache RAM blocks and the associated tag and valid RAM blocks using standard ASIC RAM compilers.

See Chapter 7 *Level 1 Memory System* for more information.

2.1.6 L2 AXI interfaces

The L2 AXI interfaces enable the L1 memory system to have access to peripherals and to external memory using an AXI master port.

See Chapter 8 *Level 2 Memory Interface* for more information.

2.1.7 Media Processing Engine

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

See the *Cortex-A5 NEON Media Processing Engine Technical Reference Manual* for more information.

2.1.8 Floating-Point Unit

The optional *Floating-Point Unit* (FPU) implements the ARMv7 VFPv4-D16 architecture and includes the VFP register file and status registers. It performs floating-point operations on the data held in the VFP register file.

See the *Cortex-A5 Floating-Point Unit Technical Reference Manual* for more information.

2.1.9 Debug

The Cortex-A5 processor has a CoreSight compliant *Advanced Peripheral Bus version 3* (APBv3) debug interface. This permits system access to debug resources, for example, the setting of watchpoints and breakpoints. The processor provides extensive support for real-time debug and performance profiling.

See Chapter 9 *Debug* for more information.

2.1.10 Performance monitoring

The Cortex-A5 processor provides performance counters and event monitors that can be configured to gather statistics on the operation of the processor and the memory system. See Chapter 10 *Performance Monitoring Unit*.

2.1.11 Virtualization extensions

The Cortex-A5 processor includes a number of ARMv7 extensions to improve the efficiency of para-virtualization. These extensions are implemented using existing fields in the *Secure Configuration Register* on page 4-45 (SCR) and a pair of new registers; the *Virtualization Control Register* on page 4-50 (VCR) and the *Virtualization Interrupt Register* on page 4-65 (VIR). The status of pending physical or virtual interrupts is reflected in the *Interrupt Status Register* on page 4-64 (ISR). The virtualization extensions in the Cortex-A5 processor are compatible with those available in Cortex-A9.

2.2 Interfaces

The Cortex-A5 processor has the following external interfaces:

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

See the *AMBA AXI Protocol Specification*, the *CoreSight Architecture Specification*, and the *CoreSight Embedded Trace Macrocell v3.5 Architecture Specification* for more information on these interfaces.

2.2.1 ETM interface

The *Embedded Trace Macrocell* (ETM) interface enables you to connect an external ETM unit to the processor for real-time code and data tracing of the core in an embedded system.

The ETM interface collects various processor signals and drives these signals from the processor. The interface runs at the full speed of the processor. The ETM interface connects directly to the external CoreSight ETM -A5 without any additional glue logic. Figure 2-2 shows the ETM interface signals.

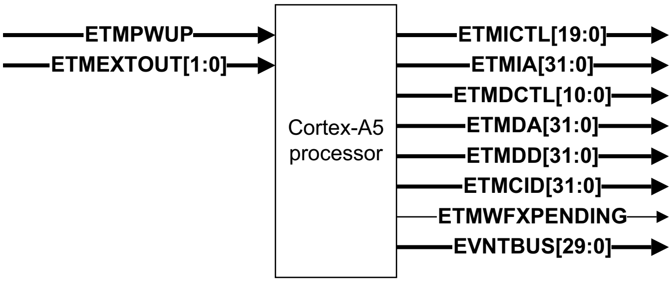


Figure 2-2 ETM interface signals

See *Trace interface signals* on page A-11 for more information.

You can disable the ETM interface to save power when not required. See the *CoreSight ETM-A5 Technical Reference Manual* for more information on the ETM interface, including the event bus, **EVNTBUS**.

2.3 Clocking and resets

The following sections describe clocking and resets:

- *Clocking*
- *Resets*.

2.3.1 Clocking

A single main clock, **CLKIN**, drives all the components of the processor. The use of a single clock domain:

- simplifies the process of clock insertion at implementation
- eliminates clock-domain crossing
- makes it simpler to support a clock grid if required.

The CPU only synchronizes the external input signals **nCPURESET**, **nIRQ**, and **nFIQ**. All other external signals must be synchronous with reference to **CLKIN**.

AXI master interface clocking

The *Bus Interface Unit* (BIU) supports AMBA AXI integer ratios through a clock enable signal **ACLKEN** (1:1, 2:1, 3:1, ...). In all cases AXI transfers remain synchronous.

Debug interface clocking

The processor includes an AMBA APB interface used to access the debug and performance monitoring registers. Internally this interface is driven from **CLKIN**. A separate enable signal, **PCLKENDBG**, is provided to enable the external AMBA bus to be driven at a lower frequency, which must be an integer ratio of **CLKIN**. If the debug infrastructure in the system is required to be fully asynchronous to the processor clock, you can use a synchronizing component supplied as part of the standard product deliverables to connect the external AMBA APB bus to the processor.

2.3.2 Resets

The Cortex-A5 processor has two reset domains with the following reset input signals:

nCPURESET This is the primary reset signal which initializes the majority of the processor except for the debug logic.

nDBGRESET This signal resets only the debug logic in the Cortex-A5 processor.

All of these are active LOW signals that reset logic in the Cortex-A5 processor.

2.4 Power management

The Cortex-A5 processor uses gated clocks and gates to disable inputs to unused functional blocks. Only the logic in use to perform an operation consumes any dynamic power.

The following sections describe power management:

- *Power control*
- *Power domains* on page 2-11
- *Communication to the Power Management Controller* on page 2-12
- *IEM support* on page 2-12.

2.4.1 Power control

The Cortex-A5 design supports four main levels of power management:

Run mode This is the normal mode of operation where all of the processor functionality is available. Everything, including core logic and embedded RAM arrays, is clocked and powered up.

Standby mode

This mode disables most of the clocks of the processor, while keeping it powered up. This reduces the power drawn to the static leakage current, plus a small clock power overhead required to enable the processor to wake up from Standby mode. The transition from Standby mode to Run mode is caused by one of the following:

- the arrival of an interrupt, either masked or unmasked
- the arrival of an event, if standby mode was initiated by a *Wait for Event* (WFE) instruction
- a debug request, when either debug is enabled or disabled
- a reset.

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

Entry into Standby mode is performed by executing the *Wait For Interrupt* (WFI) instruction. To ensure that the entry into the standby mode does not affect the memory system, the `WFI` instruction automatically performs a *Data Synchronization Barrier* (DSB) operation. This ensures that all explicit memory accesses occur in program order before the `WFI` instruction has completed.

After the processor clocks are stopped, the external **STANDBYWFI** signal is asserted to indicate that the processor is in Standby mode.

The processor can also enter Standby mode using the *Wait for Event* (WFE) instruction. In this case, the processor indicates the change of state with the external **STANDBYWFE** signal. The processor can be woken up by raising the external **EVENTI** signal on the processor.

Dormant mode

This mode is designed to enable the processor to be powered down, while leaving the TLB, instruction, and data cache powered up and maintaining their state. When the processor is in Dormant mode it can only be returned to Run mode by asserting a reset.

The mode change is initiated in the same way as Standby mode using a `WFI` instruction. When reset is asserted, the exception handler is responsible for recovering the state from storage and resuming execution of the program.

Dormant mode has an advantage over a full shutdown in that the state can be restored in a significantly shorter time with only a relatively small extra power requirement.

Before entering Dormant mode, the state of the Cortex-A5 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-A5 processor then executes a `WFI` instruction and raises **STANDBYWFI**, to indicate that it is ready to enter dormant mode. See *Communication to the Power Management Controller* on page 2-12 for more information.
- Before removing the power, the Reset signal to the Cortex-A5 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-A5 processor until the power is restored. After power is restored, the processor leaves reset and can determine that the saved state must be restored.

Shutdown mode

This mode is similar to Dormant mode, but all the power domains in the processor are shut down and all state is lost. To restore program execution after entering this mode:

- all program state must be saved and the L1 data cache cleaned
- interrupts must be disabled
- all external requests must be completed by issuing a DSB memory barrier instruction.

The processor can be restored to Run mode from reset in the same way as from Dormant mode with the exception handler restoring all the saved state.

Note

You must power up the processor before performing a reset.

Processor debug and power saving modes

The Cortex-A5 processor does not include a separate clock or power domain for the AMBA APB bus. Because of this, debug requests cannot be handled by the processor when it is in Shutdown or Dormant mode. If this functionality is required in the system, you must configure the power control infrastructure to power-up and clock the core when a debug request is issued, either by sampling the APB **PSEL** signal or using a separate mechanism.

Debug requests can be handled normally in Standby mode because the APB bus clock is gated at the same level as the interrupt logic and is always active in this mode.

2.4.2 Power domains

The Cortex-A5 processor supports two power domains:

- V_{Core}, which includes the CPU logic cells and optional ETM
- V_{Ram}, which includes the embedded memory used for the instruction and data caches and the main TLB.

Figure 2-3 shows these domains embedded in a typical *System-on-Chip* (SoC) power domain, V_{Soc}.

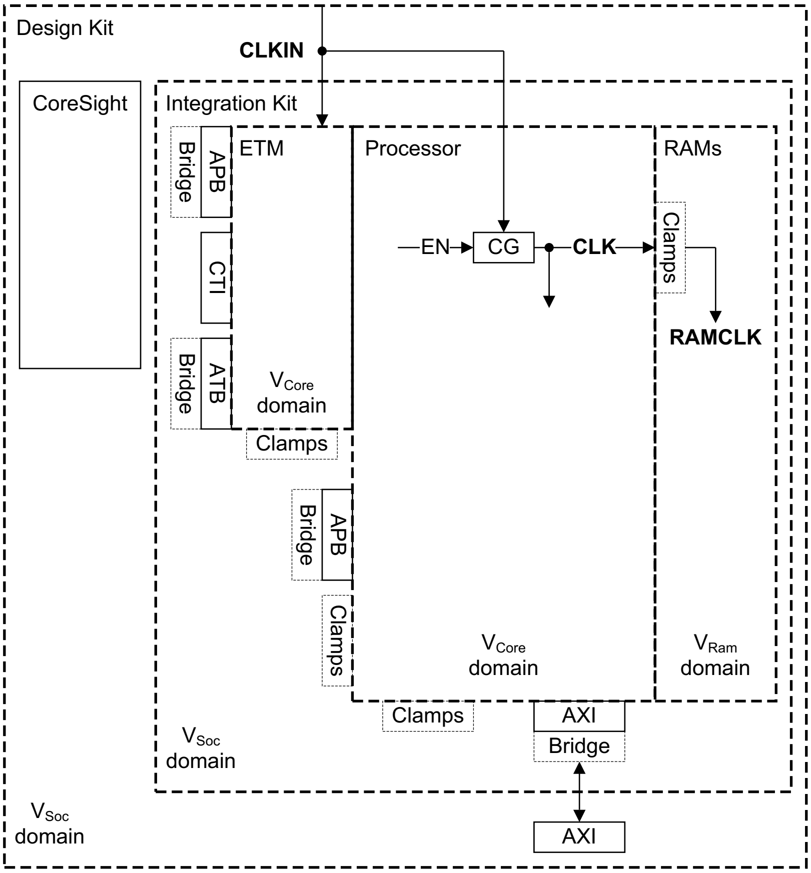


Figure 2-3 Power domains

The V_{Core} domain uses a single clock which is architecturally gated at the top level to minimize dynamic power consumption without removing power completely from the processor. The V_{Ram} domain can include logic to clamp the input signals from V_{Core}, enabling the RAM state to be retained when the power is removed from the V_{Core} domain in Dormant mode.

At the SOC integration level, the processor logic in V_{Core}, and also V_{Ram}, can be isolated and powered down completely through the instantiation of clamps and bridges on all of the external interfaces inside the V_{Soc} domain. Special bridge components supporting both asynchronous operation and power control interfaces for the AMBA AXI and APB ports on the processor and the ATB Trace port on the ETM are supplied in the integration kit provided in the standard processor deliverables. Table 2-1 shows the supported power configurations.

Table 2-1 Supported power configurations

V_{Soc}	V_{Core}	V_{Ram}	Supported?	Comments
On	Off	Off	Yes	Processor isolated from SOC domain.
On	Off	On	Yes	Dormant mode. RAM powered, clock off.
On	On	On	Yes	Run mode.

2.4.3 Communication to the Power Management Controller

Communication between the Cortex-A5 processor and the system Power Management Controller can be performed using a number of signals on the external interface. The processor can choose to enter low-power modes using the **STANDBYWFI** and **STANDBYWFE** signals. The actual power mode entered is determined by the system.

The processor includes an input clamp signal, **CPURAMCLAMP**, to enable individual domains to be isolated before the power is turned off. This must be asserted to clamp the RAM blocks for dormant mode entry.

Note

CPURAMCLAMP is only applicable if the processor has been implemented with power clamps.

2.4.4 IEM support

The Cortex-A5 processor supports the implementation of ARM *Intelligent Energy Management* (IEM) at the system level. The standard deliverables include IEM-ready asynchronous bridge components for all the AMBA AXI and APB interfaces on the processor. The ETM deliverables also include an ATB trace interface bridge component.

Chapter 3
Programmers Model

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

3.1 About the programmers model

The Cortex-A5 processor implements the ARM v7-A architecture profile. 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
- execution of 8-bit Java bytecodes
- the ThumbEE instruction set
- the security extensions, TrustZone
- the VFPv4 Floating point architecture and Advanced SIMD extensions, NEON.

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

3.2 Jazelle extension

The Cortex-A5 processor optionally provides hardware support for the Jazelle extension. The processor accelerates the execution of most Java bytecodes. Some bytecodes are executed by software routines.

See *CP14 Jazelle register summary* on page 5-3.

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.

NEON technology includes both Advanced SIMD (*single-instruction multiple data*) instructions and ARM VFPv4 instructions. These instructions are all available in both ARM and Thumb states.

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

See the *Cortex-A5 NEON Media Processing Engine Technical Reference Manual* for implementation-specific information.

3.4 Processor operating states

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

ARM state The processor executes 32-bit, word-aligned ARM instructions.

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

ThumbEE state The processor executes a variant of the Thumb instruction set designed as a target for dynamically generated code. This is code compiled on the device either shortly before or during execution from a portable bytecode or other intermediate or native representation.

Jazelle state The processor executes variable length, byte-aligned Java bytecodes.

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

Table 3-1 CPSR J and T bit encoding

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

Note

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

3.4.1 Switching state

You can change the instruction set state of the processor between:

- ARM state and Thumb state using the `BX` and `BLX` instructions.
- 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.

See the *ARM Architecture Reference Manual* for more information about changing instruction set state.

The processor handles exceptions in ARM state or in Thumb state, determined by the SCR.TE bit. See *System Control Register* on page 4-38.

On taking an exception, the processor stores the *Current Processor Status Register* (CPSR) to the *Saved Processor Status Register* (SPSR). On exiting the exception handler, the processor transfers the SPSR value back to the CPSR, returning the processor to the original instruction set state if necessary.

3.4.2 Interworking ARM and Thumb code sequences

You can 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-A5 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 unaligned accesses.

——— **Note** ———

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

———

The processor supports mixed-endian accesses. See *Memory formats* on page 3-8.

3.6 Memory formats

The Cortex-A5 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. See the *ARM Architecture Reference Manual* for more information about the supported endianness options.

3.7 Addresses in the Cortex-A5 processor

The Cortex-A5 processor operates using *virtual addresses* (VAs). The *Memory Management Unit* (MMU) translates these VAs into the *physical addresses* (PAs) used to access the memory system. Translation tables hold the mappings between VAs and PAs.

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

When the Cortex-A5 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-A5 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.

This is an example of the address manipulation that occurs when the Cortex-A5 processor requests an instruction.

1. The Cortex-A5 processor issues the VA of the instruction as Secure or Non-secure VA accesses according to the state the processor is in.
2. The instruction cache is indexed by the bits of the VA. The MMU performs the translation table look-up in parallel with the cache access. If the processor is in the Secure state it uses the Secure translation tables, otherwise it uses the Non-secure translation tables.
3. If the protection check carried out by the MMU 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 MMU passes the PA 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 translation table entry. In Secure state, both L1 and L2 translation table walk accesses are marked as Secure, even if the first level descriptor is marked as NS.

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*.

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 4-45. 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 and application code running under that operating system, executes in Non-secure state, and the most 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 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.
7. 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 signal **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 signal, see *System control and configuration* on page 4-3.

Chapter 4
System Control

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 system control

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

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

Note

See Chapter 9 *Debug* for a description of the debug registers accessed through CP14.

In ARMv7-A the following registers have instruction set equivalents:

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

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

The following sections describe system control:

- *System control functional groups*
- *System control and configuration* on page 4-3
- *MMU control and configuration* on page 4-4
- *Cache control and configuration* on page 4-4
- *Cache Operations Registers* on page 4-4
- *System performance monitor registers* on page 4-7
- *System feature registers* on page 4-7
- *c0, Instruction set attributes registers* on page 4-8
- *c7, VA to PA operations* on page 4-8
- *c8, TLB maintenance operations* on page 4-9
- *c10, Memory region remap* on page 4-10
- *c13, Software Thread ID Registers* on page 4-11
- *c15, TLB access and attributes* on page 4-12.

4.1.1 System control functional groups

The system control coprocessor provides 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 4-7.

The system control coprocessor also controls the operation of the Security Extensions:

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

Note

In Monitor mode, the Cortex-A5 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 4-45. 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 system control and configuration registers 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

You can use the input signal, **CP15SDISABLE**, to disable write access to some of the Secure registers. See Table 4-1 on page 4-4.

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

- control of the **CP15SDISABLE** signal must remain within the SoC that implements the macrocell
- the **CP15SDISABLE** signal must be driven LOW by the SoC hardware at reset.

When **CP15SDISABLE** is asserted HIGH for the registers that Table 4-1 on page 4-4 lists, any attempt to write to the secure version of a banked register, or any nonbanked register, results in an Undefined instruction exception.

Table 4-1 shows the system registers affected by **CP15SDISABLE**.

Table 4-1 System registers affected by CP15SDISABLE

Register	Register	Instruction
SCTLR	*System Control Register* on page 4-38	`MCR p15, 0, <Rd>, c1, c0, 0`
ACTLR	*Auxiliary Control Register* on page 4-41	`MCR p15, 0, <Rd>, c1, c0, 1`
TTBR0	*Translation Table Base Register 0* on page 4-51	`MCR p15, 0, <Rd>, c2, c0, 0`
TTBCR	*Translation Table Base Control Register* on page 4-53	`MCR p15, 0, <Rd>, c2, c0, 2`
DACR	*Domain Access Control Register* on page 4-55	`MCR p15, 0, <Rd>, c3, c0, 0`
PRRR	*c10, Memory region remap* on page 4-10	`MCR p15, 0, <Rd>, c10, c2, 0`
NMRR		`MCR p15, 0, <Rd>, c10, c2, 1`
VBAR	*Vector Base Address Register* on page 4-62	`MCR p15, 0, <Rd>, c12, c0, 0`
MVBAR	*Monitor Vector Base Address Register* on page 4-63	`MCR p15, 0, <Rd>, c12, c0, 1`

4.1.3 MMU control and configuration

The MMU control and configuration registers:

- 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
- control TLB maintenance operations that can invalidate either the entire TLB or selected entries
- hold thread and process IDs.

4.1.4 Cache control and configuration

The cache control and configuration registers:

- 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.

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 by which the instruction and data caches and the translation table walks of the processor are guaranteed to see the same copy of a memory location.

For Cortex-A5 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 privileged modes 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*.

Table 4-2 shows the cache operation functions and the associated data and instruction formats for CP15 c7.

Table 4-2 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 VA to PoU.	ICIMVAU	VA	`MCR p15, 0, <Rd>, c7, c5, 1`
Invalidate entire branch predictor array.	BPIALL	SBZ	`MCR p15, 0, <Rd>, c7, c5, 6`
Invalidate VA from branch predictor array.	BPIMVA	SBZ	`MCR p15, 0, <Rd>, c7, c5, 7`
Invalidate data cache line by VA to PoC.	DCIMVAC	VA	`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 VA.	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 VA to PoU.	DCCMVAU	VA	`MCR p15, 0, <Rd>, c7, c11, 1`
Clean and invalidate data cache line by VA to PoC.	DCCIMVAC	VA	`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 invalidate entire operations apply to all cache locations.

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 4-7.

Set/way format

Figure 4-1 shows the Set/Way format you can use to specify a line in the cache that must be accessed.

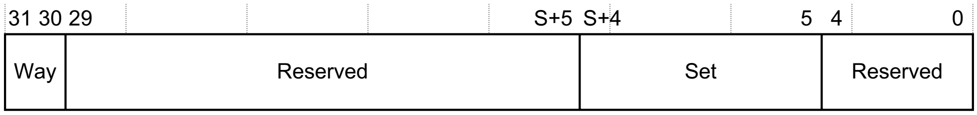


Figure 4-1 Set/Way bit assignments

Table 4-3 shows the bit assignments for Set/Way operations using CP15 c7, and their meanings.

Table 4-3 Set/Way bit assignments

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

The value of S in Table 4-3 depends on the cache size. Table 4-4 shows the relationship of cache sizes and the S parameter value.

Table 4-4 Cache size and S parameter dependency

Cache size	S parameter value
4KB	5
8KB	6
16KB	7
32KB	8
64KB	9

The value of S is derived from the following equation:

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

See *Cache Size Selection Register* on page 4-38 for details of instruction and data cache size.

Example 4-1 on page 4-7 shows the use of the command Clean Data Cache.

Example 4-1 Clean Data Cache

```
    ;code is specific to Cortex-A5 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 4-2 shows the VA bit assignments for c7 functions:

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

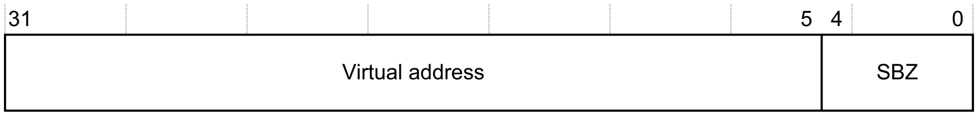


Figure 4-2 CP15 Register c7 VA bit assignments

Bits [4:0] are ignored.

The Invalidate entire instruction cache operation takes several cycles to complete and the instruction is not interruptible.

4.1.6 System performance monitor registers

The performance monitor registers control the monitoring operation and 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. See Chapter 10 *Performance Monitoring Unit* for 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 ID_MMFR0, Memory Model Feature Register 0
- Opcode_2 = 5 for ID_MMFR1, Memory Model Feature Register 1
- Opcode_2 = 6 for ID_MMFR2, Memory Model Feature Register 2
- Opcode_2 = 7 for ID_MMFR3, Memory Model Feature 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 4-30
- *Instruction Set Attributes Register 1* on page 4-31
- *Instruction Set Attributes Register 2* on page 4-32
- *Instruction Set Attributes Register 3* on page 4-33
- *Instruction Set Attributes Register 4* on page 4-34.

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 (<Rd> Field) and store the corresponding physical address in the PA Register. Figure 4-3 shows the register bit assignments.

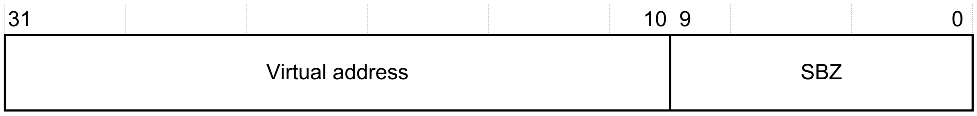


Figure 4-3 VA to PA register bit assignments

The VA to PA translation can only be performed in privileged modes 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.

The following operations are executed in either security state, and perform virtual-to-physical translations for the current security state:

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

The following operations are executed only in Secure state, and perform virtual-to-physical translations as if the core were in Non-secure state:

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

4.1.10 c8, TLB maintenance operations

The TLB Operations Register characteristics are:

Purpose Manages the TLB:
- invalidates all the entries in the TLB
- invalidates all TLB entries for an area of memory before the OS remaps it
- invalidates all TLB entries that match an ASID value.

Usage constraints Only accessible in privileged modes.

Configurations Available in all configurations.

Attributes See the register summary in Table 4-9 on page 4-14.

Table 4-5 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.

On the Cortex-A5 processor the Inner Shareable TLB maintenance operations with CRm set to c3 behave exactly the same as the instructions when CRm is set to c7, because all entries in the TLB are shareable. The ARMv7 architecture also defines TLB maintenance operations for instructions with CRm set to c5 and c6. These behave exactly the same as the instructions with Crm set to c7. Writing other Opcode_2 or CRm values is Unpredictable.

Reading from CP15 c8 is Undefined.

Table 4-5 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`
Invalidate Unified TLB by VA	TLBIMVA	VA/ASID	`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	`MCR p15, 0, <Rd>, c8, c7, 3`

Figure 4-4 shows the bit assignments of the TLB operations that use the Virtual Address as an argument. For Invalidate TLB entry by VA all ASID, the value of the ASID field is ignored.

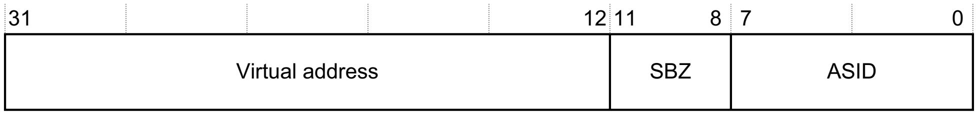


Figure 4-4 TLB Operations Register Virtual Address bit assignments

The Invalidate TLB Entries on ASID Match function requires an ASID as an argument. Figure 4-5 shows the bit assignments.

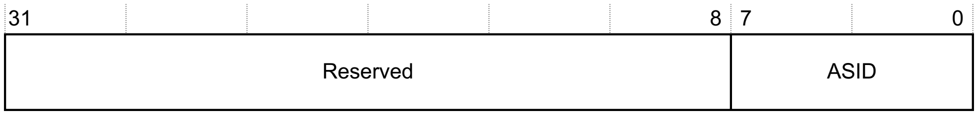


Figure 4-5 TLB Operations Register ASID bit assignments

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 entry by VA

This operation invalidates a single TLB entry which matches both the Virtual Address and ASID provided as arguments. With global entries in the TLB, the supplied ASID value is not checked.

Invalidate TLB entries by ASID match

This is a single operation that invalidates all TLB entries that match the provided ASID value. This function does not invalidate entries marked as global. In the Cortex-A5 processor, this operation takes several cycles to complete and the instruction is not interruptible.

Invalidate TLB entries by VA all ASID

You can use this Invalidate TLB Entries operation to invalidate an area of memory before remapping. You must perform an Invalidate TLB entries by VA all ASID in each area to be remapped as either section, small page, or large page. This function invalidates TLB entries that match the provided VA regardless of the ASID. The entries can be global or non-global.

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* (NMRR).

The memory region remap registers are accessed by:

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

```
MCR/MRC 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 4-6 and Table 4-7 show the encoding used for each memory type.

Table 4-6 shows the primary remapping encodings.

Table 4-6 Primary remapping encodings

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

Table 4-7 shows the normal remapping encodings.

Table 4-7 Inner or outer region type encodings

Inner or Outer Region	Encoding
Non-cacheable	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 Software Thread ID Register characteristics are:

Purpose Provide locations to store the IDs of software threads and processes for OS management purposes. These registers have no effect on processor behavior.

Usage constraints There are 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.

Configurations Available in all configurations.

Attributes See the register summary in Table 4-20 on page 4-19.

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 4-6 shows the Thread ID registers bit assignments.

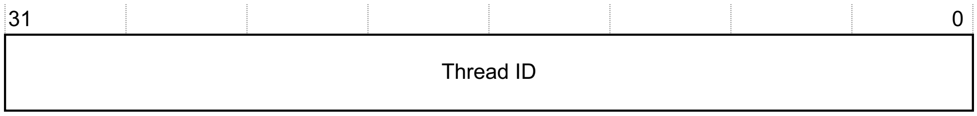


Figure 4-6 Thread ID register bit assignments

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 access and attributes

The *TLB Hitmap Register* (TLBHR) characteristics are:

Purpose Saves and restores the state of the page type hitmap held by the unified TLB.

Usage constraints Only accessible in Secure privileged modes.

Configurations Available in all configurations.

Attributes See the register summary in Table 4-21 on page 4-20.

Table 4-8 shows the TLBHR data format.

Table 4-8 TLBHR data format

Bits	Description
[31:4]	RAZ/WI.
[3]	Set if 16MB supersections are present in the TLB
[2]	Set if 1MB sections are present in the TLB
[1]	Set if 16KB pages are present in the TLB
[0]	Set if 4KB pages are present in the TLB

To access the TLBHR, use:

```
MRC p15, 5, <Rd>, c15, c0, 0; Read TLB Hitmap Register
MCR p15, 5, <Rd>, c15, c0, 0; Write TLB Hitmap Register
```

4.2 Register summary

This section contains 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 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.

Whether the corresponding exception is taken depends 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.

If the SCR.EA bit, FIQ bit, or IRQ bit is set, 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.

4.2.2 c0 summary table

Table 4-9 shows the system control registers you can access when CRn is c0.

Table 4-9 c0 system control registers

Op1	CRm	Op2	Name	Type	Reset	Description
0	c0	0	MIDR	RO	`0x410FC051`	*Main ID Register* on page 4-21
		1	CTR	RO	`0x83338003`	*Cache Type Register* on page 4-21
		2	TCMTR	RO	`0x00000000`	*TCM Type Register* on page 4-22
		3	TLBTR	RO	`0x00000000`	*TLB Type Register* on page 4-23
		4	-	-	-	-
		5	MPIDR	RO	-[a]	*Multiprocessor Affinity Register* on page 4-23
		6-7	-	-	-	-
	c1	0	ID_PFR0	RO	`0x00001231`	*Processor Feature Register 0* on page 4-24
		1	ID_PFR1	RO	`0x00000011`	*Processor Feature Register 1* on page 4-24
		2	ID_DFR0	RO	`0x02010444`	*Debug Feature Register 0* on page 4-25
		3	Reserved	-	-	-
		4	ID_MMFR0	RO	`0x00100003`	*Memory Model Features Register 0* on page 4-26
		5	ID_MMFR1	RO	`0x40000000`	*Memory Model Features Register 1* on page 4-27
		6	ID_MMFR2	RO	`0x01230000`	*Memory Model Features Register 2* on page 4-28
		7	ID_MMFR3	RO	`0x00102211`	*Memory Model Features Register 3* on page 4-29
	c2	0	ID_ISAR0	RO	`0x00101111`	*Instruction Set Attributes Register 0* on page 4-30
		1	ID_ISAR1	RO	`0x13112111`	*Instruction Set Attributes Register 1* on page 4-31
		2	ID_ISAR2	RO	`0x21232041`	*Instruction Set Attributes Register 2* on page 4-32
		3	ID_ISAR3	RO	`0x11112131`	*Instruction Set Attributes Register 3* on page 4-33
		4	ID_ISAR4	RO	`0x00011142`	*Instruction Set Attributes Register 4* on page 4-34
		5	ID_ISAR5	RO	-	*Instruction Set Attributes Register 5* on page 4-35
		6-7	Reserved	-	-	-
	c3-c7	0-7	Reserved	-	-	-
1	c0	0	CCSIDR	RO	-	*Cache Size Identification Register* on page 4-35
		1	CLIDR	RO	`0x09200003`	*Cache Level ID Register* on page 4-36
		7	AIDR	RO	`0x00000000`	*Auxiliary ID Register* on page 4-37
2	c0	0	CSSELR	RW	-	*Cache Size Selection Register* on page 4-38
3-7	c0-c15	0-7	-	-	-	-

[a] Dependent on external signal **CLUSTERID**.

4.2.3 c1 summary table

Table 4-10 shows the system control registers you can access when CRn is c1.

Table 4-10 c1 system control registers

Op1	CRm	Op2	Name	Type	Reset	Description
0	c0	0	SCTLR	RW	`0x00C50078`	*System Control Register* on page 4-38
		1	ACTLR[a]	RW	`0x00006000`	*Auxiliary Control Register* on page 4-41
		2	CPACR	RW	-[b]	*Coprocessor Access Control Register* on page 4-43
	c1	0	SCR[c]	RW	`0x00000000`	*Secure Configuration Register* on page 4-45
		1	SDER[c]	RW	`0x00000000`	*Secure Debug Enable Register* on page 4-47
		2	NSACR	RW[d]	-[e]	*Non-secure Access Control Register* on page 4-48
		3	VCR[c]	RW	`0x00000000`	*Virtualization Control Register* on page 4-50

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

b. `0x00000000` if NEON present, `0xC0000000` if FPU present, and `0x00000000` for integer only.

c. No access in Non-secure state. See *Virtualization* on page 4-13.

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, `0x0000C000` if FPU present, and `0x00000000` for integer only.

4.2.4 c2 summary table

Table 4-11 shows the system control registers you can access when CRn is c2.

Table 4-11 c2 system control registers

Op1	CRm	Op2	Name	Type	Reset	Description
0	c0	0	TTBR0	RW	-	*Translation Table Base Register 0* on page 4-51
		1	TTBR1	RW	-	*Translation Table Base Register 1* on page 4-52
		2	TTBCR	RW	`0x00000000`[a]	*Translation Table Base Control Register* on page 4-53

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

4.2.5 c3 summary table

Table 4-12 shows the system control register you can access when CRn is c3.

Table 4-12 c3 system control register

Op1	CRm	Op2	Name	Type	Reset	Description
0	c0	0	DACR	RW	-	*Domain Access Control Register* on page 4-55

4.2.6 c4 summary table

There are no system control registers when CRn is c4.

4.2.7 c5 summary table

Table 4-13 shows the system control registers you can access when CRn is c5.

Table 4-13 c5 system control registers

Op1	CRm	Op2	Name	Type	Reset	Description
0	c0	0	DFSR	RW	-	*Data Fault Status Register* on page 4-56
		1	IFSR	RW	-	*Instruction Fault Status Register* on page 4-57
	c1	0	ADFSR	-	-	*Auxiliary Data Fault Status Register* on page 4-59
		1	AIFSR	-	-	*Auxiliary Instruction Fault Status Register* on page 4-59

4.2.8 c6 summary table

Table 4-14 shows the system control registers you can access when CRn is c6.

Table 4-14 c6 system control registers

Op1	CRm	Op2	Name	Type	Reset	Description
0	c0	0	DFAR	RW	-	*Data Fault Address Register* on page 4-60
		2	IFAR	RW	-	*Instruction Fault Address Register* on page 4-60

4.2.9 c7 summary table

Table 4-15 shows the system control registers you can access when CRn is c7.

Table 4-15 c7 system control registers

Op1	CRm	Op2	Name	Type	Reset	Description
0	c0	4	NOP	WO	-	-
	c1	0	ICIALLUIS	WO	-	*Cache Operations Registers* on page 4-4
		6	BPIALLIS	WO	-	-
	c4	0	PAR	RW	-	*Physical Address Register* on page 4-60
	c5	0	ICIALLU	WO	-	*Cache Operations Registers* on page 4-4
		1	ICIMVAU	WO	-	
		4	ISB	WO	-	-
		6	BPIALL	WO	-	*Cache Operations Registers* on page 4-4
		7	BPIMVA	WO	-	
	c6	1	DCIMVAC	WO	-	
		2	DCISW	WO	-	
	c8	0-3	V2PCWPR	WO	-	*c7, VA to PA operations* on page 4-8
		4-7	V2POWPR	WO	-	

Table 4-15 c7 system control registers (continued)

Op1	CRm	Op2	Name	Type	Reset	Description
0	c10	1	DCCMVAC	WO	-	*Cache Operations Registers* on page 4-4
		2	DCCSW	WO	-	
		4	DSB	WO	-	-
		5	DMB	WO	-	-
	c11	1	DCCMVAU	WO	-	*Cache Operations Registers* on page 4-4
	c14	1	DCCIMVAC	WO	-	
		2	DCCISW	WO	-	

4.2.10 c8 summary table

Table 4-16 shows the system control registers you can access when CRn is c8.

Table 4-16 c8 system control register

Op1	CRm	Op2	Name	Type	Reset	Description
0	c3	0	TLBIALLIS	WO	-	*c8, TLB maintenance operations* on page 4-9
		1	TLBIMVAIS	WO	-	
		2	TLBIASIDIS	WO	-	
		3	TLBIMVAAIS	WO	-	
	c5, c6, or c7	0	TLBIALL	WO	-	
		1	TLBIMVA	WO	-	
		2	TLBIASID	WO	-	
		3	TLBIMVAA	WO	-	

4.2.11 c9 summary table

Table 4-17 shows the system control registers you can access when CRn is c9.

Table 4-17 c9 system control registers

Op1	CRm	Op2	Name	Type	Reset	Description
0	c12	0	PMCR	RW	`0x41052000`	*Performance Monitor Control Register* on page 10-3
		1	PMCNTENSET	RW	`0x00000000`	*Count Enable Set Register* on page 10-4
		2	PMCNTENCLR	RW	`0x00000000`	*Count Enable Clear Register* on page 10-5
		3	PMOVSR	RW	-	*Overflow Flag Status Register* on page 10-6
		4	PMSWINC	WO	-	*Software Increment Register* on page 10-7
		5	PMSELR	RW	`0x00000000`	*Event Counter Selection Register* on page 10-8
		6	PMCEID0	RO	`0x003FFFFF`	*Common Event Identification Registers* on page 10-9
		7	PMCEID1	RO	-	
	c13	0	PMCCNTR	RW	-	*Cycle Count Register* on page 10-10
		1[a]	PMXEVTYPER	RW	-	*Event Type Select Register* on page 10-11
			PMCCFILTR	RW	-	*Cycle Count Filter Control Register* on page 10-13
		2	PMXEVCNTR	RW	-	*Event Count Registers* on page 10-14
	c14	0	PMUSERENR	RW[b]	`0x00000000`	*User Enable Register* on page 10-15
		1	PMINTENSET	RW	`0x00000000`	*Interrupt Enable Set Register* on page 10-16
		2	PMINTENCLR	RW	`0x00000000`	*Interrupt Enable Clear Register* on page 10-16

a. Select the use of these registers in the *Event Counter Selection Register* on page 10-8.

b. RO in user mode

Note

See Chapter 10 *Performance Monitoring Unit* for more information on system performance monitoring.

4.2.12 c10 summary table

Table 4-18 shows the system control registers you can access when CRn is c10.

Table 4-18 c10 system control registers

Op1	CRm	Op2	Name	Type	Reset	Description
0	c2	0	PRRR	RW	`0x00098AA4`	*c10, Memory region remap* on page 4-10
		1	NMRR	RW	`0x44E048E0`	

4.2.13 c11 summary table

There are no system control registers when CRn is c11.

4.2.14 c12 summary table

Table 4-19 shows the system control registers you can access when CRn is c12.

Table 4-19 c12 system control registers

Op1	CRm	Op2	Name	Type	Reset	Description
0	c0	0	VBAR	RO	-	*Vector Base Address Register* on page 4-62
		1	MVBAR	RO	-	*Monitor Vector Base Address Register* on page 4-63
	c1	0	ISR	RO	-	*Interrupt Status Register* on page 4-64
		1	VIR	RW	`0x00000000`	*Virtualization Interrupt Register* on page 4-65

4.2.15 c13 summary table

Table 4-20 shows the system control registers you can access when CRn is c13.

Table 4-20 c13 system control registers

Op1	CRm	Op2	Name	Type	Reset	Description
0	c0	0	FCSEIDR	RO	`0x00000000`	*Fast Context Switch Extension* (FCSE) not implemented
		1	CONTEXTIDR	RW	-	*Context ID Register* on page 4-66
		2	TPIDRURW	RW[a]	-	*c13, Software Thread ID Registers* on page 4-11
		3	TPIDRURO	RO[b]	-	
		4	TPIDRPRW.	RW	-	

a. RW in User mode

b. RO in User mode

4.2.16 c14 summary table

There are no system control registers when CRn is c14.

4.2.17 c15 summary table

Table 4-21 on page 4-20 shows the system control registers you can access when CRn is c15.

Table 4-21 c15 system control registers

Op1	CRm	Op2	Name	Type	Reset	Description
3[a]	c0	0	Data Register 0	RO	-	-
		1	Data Register 1	RO	-	
	c2	0	Data Cache Tag Read Operation Register	WO	-	
		1	Instruction Cache Tag Read Operation Register	WO	-	
	c4	0	Data Cache Data Read Operation Register	WO	-	
		1	Instruction Cache Data Read Operation Register	WO	-	
		2	TLB Data Read Operation Register	WO	-	
4	c0	0	Configuration Base Address	RO[b]	[c]	*Configuration Base Address Register* on page 4-66
5	c0	0	TLB Hitmap	RW	-	*c15, TLB access and attributes* on page 4-12

a. See *Direct access to internal memory* on page 7-8 for information on how these registers are used.
b. *Write Ignored* (WI) in Privileged modes and Undefined in User mode.
c. The configuration base address is set to zero.

4.3 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:

- 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 4-9 on page 4-14.

Figure 4-7 shows the MIDR bit assignments.

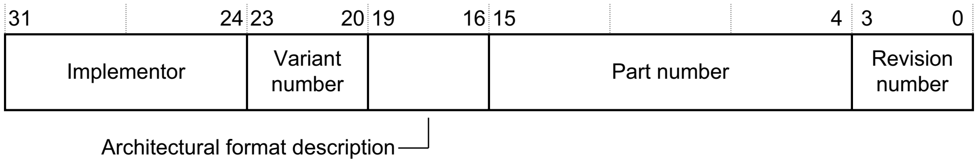


Figure 4-7 MIDR bit assignments

Table 4-22 shows the MIDR bit assignments.

Table 4-22 MIDR bit assignments

Bits	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.
[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.

See the reset value in Table 4-9 on page 4-14 for the field values.

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

- 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 4-9 on page 4-14.

Figure 4-8 shows the CTR bit assignments.

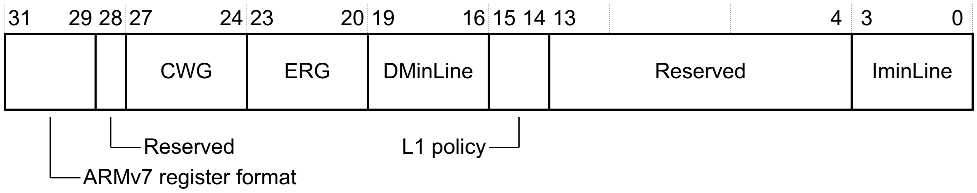


Figure 4-8 CTR bit assignments

Table 4-23 shows the CTR bit assignments.

Table 4-23 CTR bit assignments

Bits	Name	Description
[31:29]	-	ARMv7 register format.
[28]	Reserved	RAZ.
[27:24]	CWG	Cache Writeback Granule. Log_2 of the number of words of the maximum size of memory that can be overwritten as a result of the eviction of a cache entry that has had a memory location in it modified. b0011 = 8 word granularity for the Cortex-A5 processor.
[23:20]	ERG	Exclusives Reservation Granule. Log_2 of the number of words of the maximum size of the reservation granule that has been implemented for the Load-Exclusive and Store-Exclusive instructions. b0011 = 8 word granularity for the Cortex-A5 processor.
[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. b0011 = 8 words for the Cortex-A5 processor.
[15:14]	L1 policy	Indicates the level 1 instruction cache policy for indexing and tagging. b10 virtual index, physical tag, VIPT.
[13:4]	Reserved	RAZ.
[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. b0011 = 8 words for the Cortex-A5 processor.

To access the CTR, use:

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

4.3.3 TCM Type Register

The Cortex-A5 processor does not implement instruction or data *Tightly Coupled Memory* (TCM), so this register always reads-as-zero (RAZ).

4.3.4 TLB Type Register

The *Translation Lookaside Buffer* (TLB) Type Register, TLBTR, returns the number of lockable entries for the TLB. The Cortex-A5 processor does not implement this feature, so this register always RAZ.

4.3.5 Multiprocessor Affinity Register

The MPIDR characteristics are:

Purpose To identify:

- whether the processor is part of a Cortex-A5 MPCore implementation
- the target core in a Cortex-A5 MPCore processor.

Usage constraints The MPIDR is:

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

Configurations Available in all configurations. The value of bit [30] indicates a Cortex-A5 MPCore multiprocessor or a uniprocessor configuration.

Attributes See the register summary in Table 4-9 on page 4-14.

Figure 4-9 shows the MPIDR bit assignments.

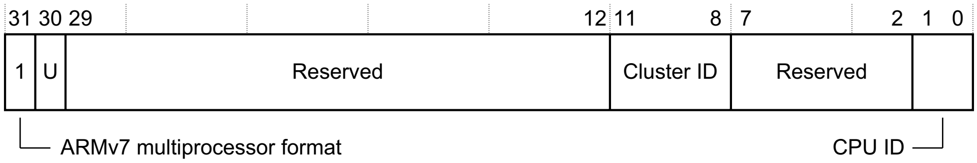


Figure 4-9 MPIDR bit assignments

Table 4-24 shows the MPIDR bit assignments.

Table 4-24 MPIDR bit assignments

Bits	Name	Description
[31]	Reserved	Indicates that the register uses the new multiprocessor format. This is always 1.
[30]	U bit	1 = Processor is always part of a uniprocessor system.
[29:12]	Reserved	SBZ.
[11:8]	Cluster ID	Value read in **CLUSTERID** configuration pins. It identifies a Cortex-A5 processor in a system with more than one Cortex-A5 processor present.
[7:2]	Reserved	SBZ.
[1:0]	CPU ID	The value depends on the number of configured processors. For the Cortex-A5 processor, the CPU ID is `0x0`.

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 4-9 on page 4-14.

Figure 4-10 shows the ID_PFR0 bit assignments.

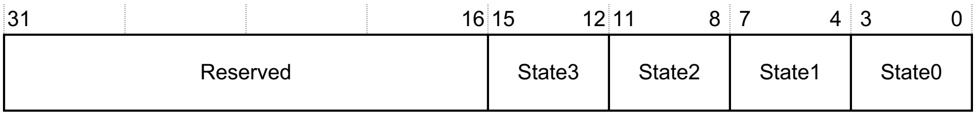


Figure 4-10 ID_PFR0 bit assignments

Table 4-25 shows the ID_PFR0 bit assignments.

Table 4-25 ID_PFR0 bit assignments

Bits	Name	Description
[31:16]	Reserved	RAZ.
[15:12]	State3	`0x1` Jazelle RCT 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:

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

The ID_PFR1 must be interpreted with ID_PFR0.

Configurations Available in all configurations.

Attributes See the register summary in Table 4-9 on page 4-14.

Figure 4-11 shows the ID_PFR1 Register bit assignments.

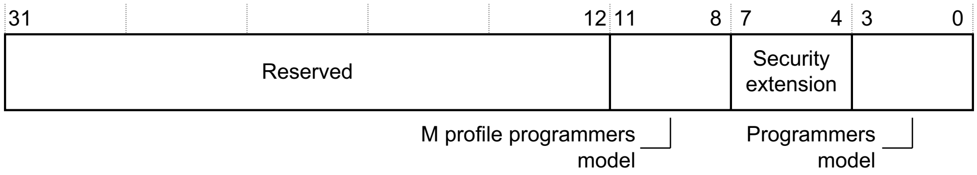


Figure 4-11 ID_PFR1 bit assignments

Table 4-26 shows the ID_PFR1 bit assignments.

Table 4-26 ID_PFR1 bit assignments

Bits	Name	Description
[31:12]	Reserved	RAZ.
[11:8]	M profile programmers model	`0x0` Not supported.
[7:4]	Security extensions	`0x1` Security extension architecture v1 supported.
[3:0]	Programmers model	`0x1` standard ARMv4 programmers model supported.

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 4-9 on page 4-14.

Figure 4-12 shows the ID_DFR0 bit assignments.

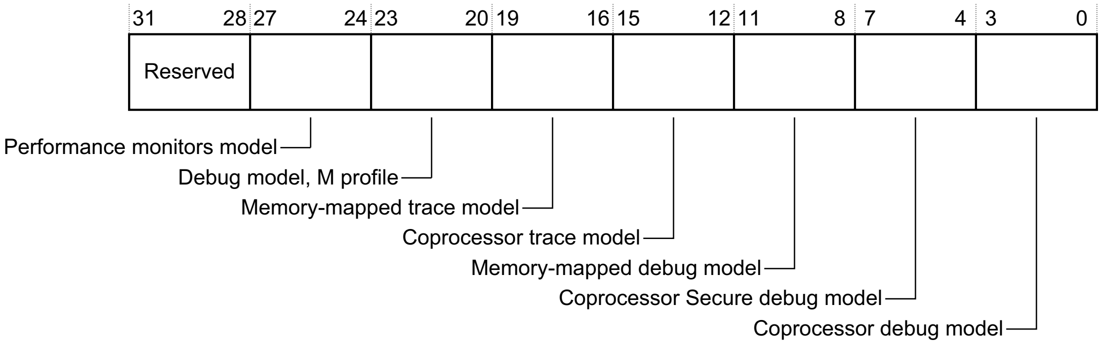

31
28
27
24
23
20
19
16
15
12
11
8
7
4
3
0
Reserved
Performance monitors model
Debug model, M profile
Memory-mapped trace model
Coprocessor trace model
Memory-mapped debug model
Coprocessor Secure debug model
Coprocessor debug model

Figure 4-12 ID_DFR0 bit assignments

Table 4-27 shows the ID_DFR0 bit assignments.

Table 4-27 ID_DFR0 bit assignments

Bits	Name	Description
[31:28]	Reserved	RAZ.
[27:24]	Performance monitors model	`0x2` Version 2.
[23:20]	Debug model, M profile	`0x0` Not supported.
[19:16]	Memory-mapped trace model	`0x1` Memory-mapped trace debug model supported.
[15:12]	Coprocessor trace model	`0x0` Not supported.
[11:8]	Memory-mapped debug model	`0x4` Memory-mapped core debug model supported.
[7:4]	Coprocessor Secure debug model	`0x4` Coprocessor based secure debug model supported using CP14.
[3:0]	Coprocessor debug model	`0x4` Coprocessor based core debug model supported supported 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 4-28 shows the ID_AFR0 bit assignments.

Table 4-28 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 4-9 on page 4-14.

Figure 4-13 on page 4-27 shows the ID_MMFR0 bit assignments.

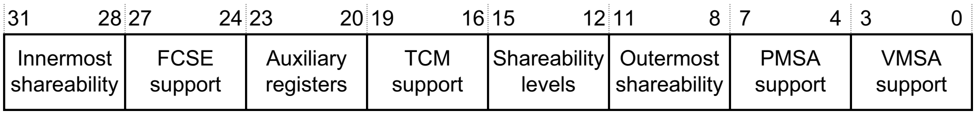


Figure 4-13 ID_MMFR0 bit assignments

Table 4-29 shows the ID_MMFR0 bit assignments.

Table 4-29 ID_MMFR0 bit assignments

Bits	Name	Description
[31:28]	Innermost shareability	`0x0` Not supported.
[27:24]	FCSE support	`0x0` Not supported.
[23:20]	Auxiliary registers	`0x1` One Auxiliary Control Register supported.
[19:16]	TCM support	`0x0` Not supported.
[15:12]	Shareability levels	`0x0` One level of shareability supported.
[11:8]	Outermost shareability	`0x0` L1 cache coherency not supported.
[7:4]	PMSA support	`0x0` Not supported.
[3:0]	VMSA support	`0x3` *Virtual Memory System Architecture* (VMSA) supported including remapping of access permission flags.

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 4-9 on page 4-14.

Figure 4-14 shows the ID_MMFR1 bit assignments.

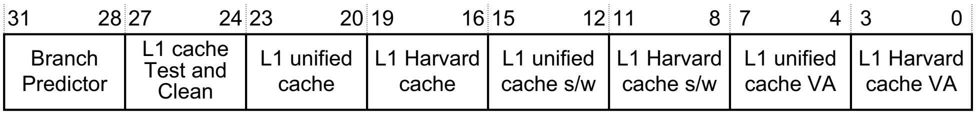


Figure 4-14 ID_MMFR1 bit assignments

Table 4-30 shows the ID_MMFR1 bit assignments.

Table 4-30 ID_MMFR1 bit assignments

Bits	Name	Description
[31:28]	Branch Predictor	`0x4` For execution correctness, Branch Predictor requires no flushing at any time.
[27:24]	L1 cache Test and Clean	`0x0` L1 data cache test and clean operation not supported.
[23:20]	L1 unified cache	`0x0` L1 unified cache clean/invalidate-all operation not supported.
[19:16]	L1 Harvard cache	`0x0` L1 data cache clean/invalidate-all operation not supported.
[15:12]	L1 unified cache s/w	`0x0` L1 unified cache maintenance operations by set/way not supported.
[11:8]	L1 Harvard cache s/w	`0x0` L1 Harvard cache maintenance operations by set/way not supported.
[7:4]	L1 unified cache VA	`0x0` L1 unified cache maintenance operations by VA not supported.
[3:0]	L1 Harvard cache VA	`0x0` L1 Harvard cache maintenance operations by VA 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 4-9 on page 4-14.

Figure 4-15 shows the ID_MMFR2 bit assignments.

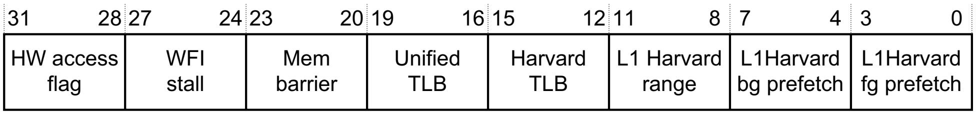


Figure 4-15 ID_MMFR2 bit assignments

Table 4-31 shows the ID_MMFR2 bit assignments.

Table 4-31 ID_MMFR2 bit assignments

Bits	Name	Description
[31:28]	HW access flag	`0x0` Not supported.
[27:24]	WFI stall	`0x1` *Wait For Interrupt* (WFI) supported.
[23:20]	Mem barrier	`0x2` Supports: • *Data Synchronization Barrier* (DSB) • *Instruction Synchronization Barrier* (ISB) • *Data Memory Barrier* (DMB).
[19:16]	Unified TLB	`0x3` Supports: • invalidate all entries • invalidate TLB entry by VA • invalidate TLB entries by ASID match.
[15:12]	Harvard TLB	`0x0` Not supported.
[11:8]	L1 Harvard range	`0x0` Not supported.
[7:4]	L1 Harvard background prefetch	`0x0` Not supported.
[3:0]	L1 Harvard foreground prefetch	`0x0` 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 4-9 on page 4-14.

Figure 4-16 shows the ID_MMFR3 bit assignments.

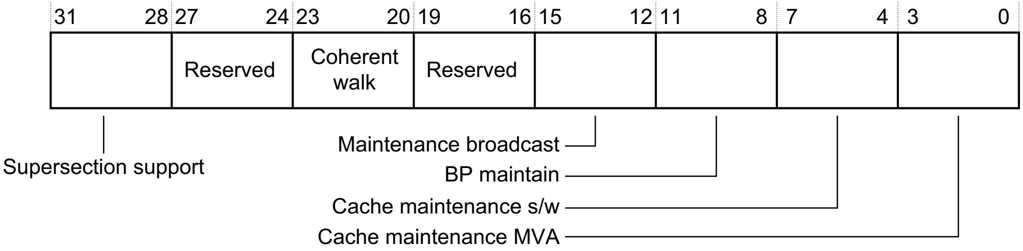


Figure 4-16 ID_MMFR3 bit assignments

Table 4-32 shows the ID_MMFR3 bit assignments.

Table 4-32 ID_MMFR3 bit assignments

Bits	Name	Description
[31:28]	Supersection support	`0x0` 16MB supersections supported.
[27:24]	Reserved	RAZ.
[23:20]	Coherent walk	`0x1` Supported. Updates to translation tables do not require a clean to the point of unification to ensure visibility by subsequent translation table walks.
[19:16]	Reserved	RAZ.
[15:12]	Maintenance broadcast	`0x2` Supported. Cache, TLB and Branch prediction operations broadcast.
[11:8]	BP maintain	`0x2` Supports: • invalidate entire branch predictor array • invalidate branch predictor by MVA.
[7:4]	Cache maintenance set/way	`0x1` Supported.
[3:0]	Cache maintenance MVA	`0x1` 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 4-9 on page 4-14.

Figure 4-17 shows the ID_SAR0 bit assignments.

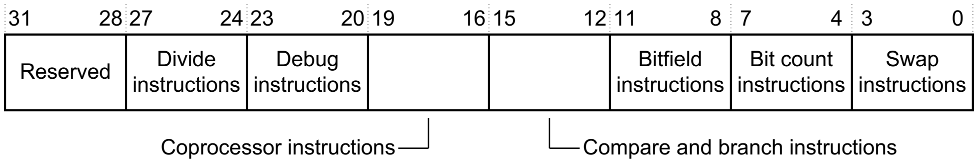


Figure 4-17 ID_ISAR0 bit assignments

Table 4-33 shows the ID_ISAR0 bit assignments.

Table 4-33 ID_ISAR0 bit assignments

Bits	Name	Description
[31:28]	Reserved	RAZ.
[27:24]	Divide instructions	`0x0` Integer hardware divide not supported.
[23:20]	Debug instructions	`0x1` BKPT supported.
[19:16]	Coprocessor instructions	`0x0` Not supported. VFPv4, CP14, and CP15 described elsewhere.
[15:12]	Compare and branch instructions	`0x1` CBZ and CBNZ supported.
[11:8]	Bit field instructions	`0x1` BFC, BFI, SBFX, and UBFX supported.
[7:4]	Bit count instructions	`0x1` CLZ supported.
[3:0]	Swap instructions	`0x1` SWP and SWPB supported.

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 4-9 on page 4-14.

Figure 4-18 shows the ID_ISAR1 bit assignments.

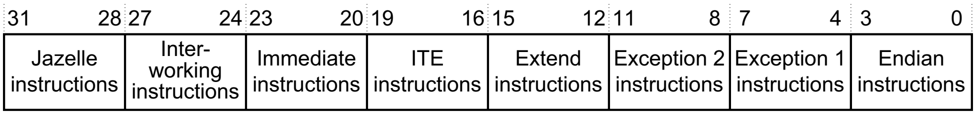


Figure 4-18 ID_ISAR1 bit assignments

Table 4-34 shows the ID_ISAR1 bit assignments.

Table 4-34 ID_ISAR1 bit assignments

Bits	Name	Description
[31:28]	Jazelle instructions	`0x1` BXJ and J bit in PSRs supported.
[27:24]	Interwork instructions	`0x3` ARM/Thumb interworking instructions supported. `BX`, `BLX`, T bit in PSRs PC loads and `DP` instructions have `BX`-like behavior
[23:20]	Immediate instructions	`0x1` Special immediate-generating instructions supported.

Table 4-34 ID_ISAR1 bit assignments (continued)

Bits	Name	Description
[19:16]	IfThen instructions	`0x1` Thumb IT blocks supported.
[15:12]	Extend instructions	`0x2` All supported
[11:8]	Exception2 instructions	`0x1` `SRS`, `RFE`, and `CPS` supported.
[7:4]	Exception1 instructions	`0x1` (`LDM(2)`, `LDM(3)`, and `STM(2)` supported.
[3:0]	Endian instructions	`0x1` BE8 endian change supported with `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 4-9 on page 4-14.

Figure 4-19 shows the ID_ISAR2 bit assignments.

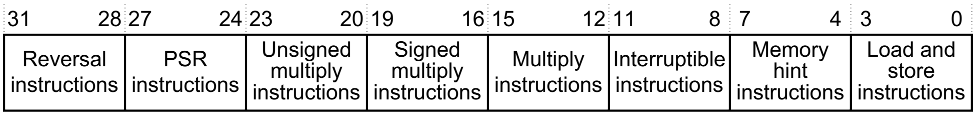


Figure 4-19 ID_ISAR2 bit assignments

Table 4-35 shows the ID_SAR2 bit assignments.

Table 4-35 ID_ISAR2 bit assignments

Bits	Name	Description
[31:28]	Reversal instructions	`0x2` Reversal instructions `REV`, `REV16`, `REVSH`, and `RBIT` supported.
[27:24]	PSR instructions	`0x1` PSR instructions `MRS` and `MSR`, and exception return data-processing instructions supported.
[23:20]	Multiply instructions (advanced, unsigned)	`0x2` Long multiply instructions and `UMAAL` supported.
[19:16]	Multiply instructions (advanced, signed)	`0x3` Multiply instructions `SMULL`, `SMAL`, `SMLABB`, `SMLABT`, `SMLALBB`, `SMLALBT`, `SMLALTB`, `SMLALTT`, `SMLATB`, `SMLATT`, `SMLAWB`, `SMLAWT`, `SMULBB`, `SMULBT`, `SMULTB`, `SMULTT`, `SMULWB`, `SMULWT`, and Q flag in PSRs supported.
[15:12]	Multiply instructions	`0x2` Multiply instructions `MUL`, `MLA`, and `MLS` supported.

Table 4-35 ID_ISAR2 bit assignments (continued)

Bits	Name	Description
[11:8]	Multi-access interruptible instructions	`0x0` Interrupting of multi-cycle operations not supported.
[7:4]	Memory hint instructions	`0x4` memory hint instructions `PLD`, `PLI`, and `PLDW` supported.
[3:0]	Load and store instructions	`0x1` Load and store instructions `LDRD` or `STRD` supported.

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 4-9 on page 4-14.

Figure 4-20 shows the ID_ISAR3 bit assignments.

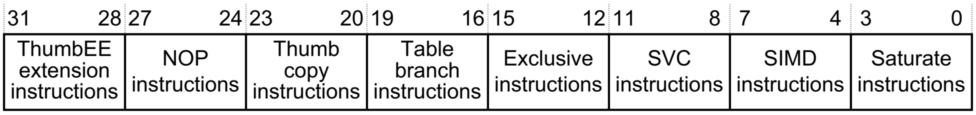


Figure 4-20 ID_ISAR3 bit assignments

Table 4-36 shows the ID_ISAR3 bit assignments.

Table 4-36 ID_ISAR3 bit assignments

Bits	Name	Description
[31:28]	Thumb-2 Executable Environment Extension instructions	`0x1` Thumb-2 Executable Environment Extension instructions supported
[27:24]	True NOP instructions	`0x1` True no-operation instruction NOP32 supported.
[23:20]	Thumb copy instructions	`0x1` Thumb copy instruction Thumb `MOV(3)` low reg -> low reg and `CPY` alias supported.
[19:16]	Table branch instructions	`0x1` Thumb table branch instruction `TBB` and `TBH` supported
[15:12]	Exclusive instructions	`0x2` Exclusive instructions `LDREX`, `STREX`, `LDRBEX`, `STRBEX`, `LDRHEX`, `STRHEX`, `LDRDEX`, `STRDEX`, and `CLREX` supported.
[11:8]	SVC instructions	`0x1` Supported.
[7:4]	SIMD instructions	`0x3` All supported.
[3:0]	Saturate instructions	`0x1` Saturate instructions `QADD`, `QDADD`, `QDSUB`, and `QSUB`, and Q flag in PSRs supported.

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 4-9 on page 4-14.

Figure 4-21 shows the ID_ISAR4 bit assignments.

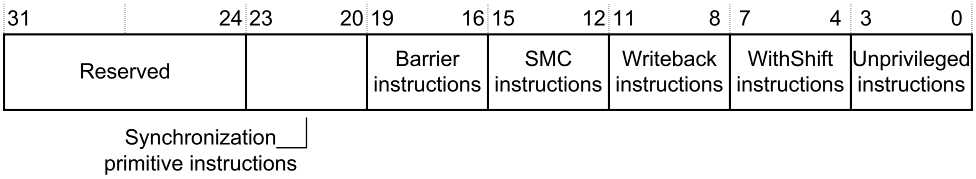


Figure 4-21 ID_ISAR4 bit assignments

Table 4-37 shows the ID_ISAR4 bit assignments.

Table 4-37 ID_ISAR4 bit assignments

Bits	Name	Description
[31:24]	Reserved	RAZ.
[23:20]	Synchronization primitive instructions	`0x0` Synchronization primitive instructions supported.
[19:16]	Barrier instructions	`0x1` Barrier instructions `DMB`, `DSB`, and `ISB` supported.
[15:12]	SMC instructions	`0x1` *Secure Monitor Call* (`SMC`) instructions supported.
[11:8]	Write-back instructions	`0x1` All defined write-back addressing modes supported.
[7:4]	With-shift instructions	`0x4` Immediate and register control shifted operations supported: • shifts of loads and stores over the range LSL 0-3 • constant shift options • register-controlled shift options.
[3:0]	Unprivileged instructions	`0x2` Unprivileged load/store instructions `LDRT` and `STRT` supported.

To access the ID_ISAR4 use:

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

4.3.19 Instruction Set Attributes Register 5

The ID_ISAR5 characteristics are:

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

Usage constraints The ID_ISAR5 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 4-9 on page 4-14.

Table 4-38 shows the ID_ISAR5 bit assignments.

Table 4-38 ID_ISAR5 bit assignments

Bits	Name	Description
[31:0]	Reserved	RAZ.

To access the ID_ISAR5 use:

```
MRC p15, 0, <Rd>, c0, c2, 5
```

4.3.20 Instruction Set Attributes Registers 6-7

ID_ISAR6 and ID_ISAR7 are reserved, and read as `0x00000000`.

4.3.21 Cache Size Identification Register

The CCSIDR characteristics are:

Purpose Provides information about the architecture of the caches selected by the *Cache Size Selection Register* on page 4-38.

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 4-9 on page 4-14.

Figure 4-22 shows the CCSIDR bit assignments.

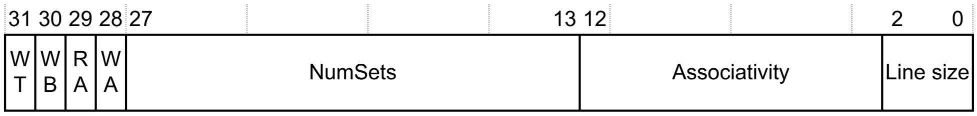


Figure 4-22 CCSIDR bit assignments

Table 4-39 shows the CSSIDR bit assignments.

Table 4-39 CCSIDR bit assignments

Bits	Name	Description		
[31]	WT	Write-Through not supported. Read as `0x0`.		
[30]	WB	Write-Back supported only for L1 data cache. Read as: `0x1` for data cache `0x0` for instruction cache.		
[29]	RA	Cache read allocation supported. Read as `0x1`.		
[28]	WA	Write allocation supported only for L1 data cache. Read as: `0x1` for data cache `0x0` for instruction cache.		
[27:13]	NumSets	Indicates number of cache sets. The number of sets depends on the cache size and whether data or instruction cache is selected using the Cache Size Selection Register:		
		Cache size	Data	Instruction
		4KB	`0x1F`	`0x3F`
		8KB	`0x3F`	`0x7F`
		16KB	`0x7F`	`0xFF`
		32KB	`0xFF`	`0x1FF`
		64KB	`0x1FF`	`0x3FF`
[12:3]	Associativity	Indicates cache associativity. Read as: `0x3` for 4-way data cache `0x1` for 2-way instruction cache.		
[2:0]	LineSize	Indicates number of words per line. `0x1` = 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, bits[31:28] are b0010.

If the CSSELR reads the data cache values, bits[31:28] are b0111. See *Cache Size Selection Register* on page 4-38.

4.3.22 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 4-9 on page 4-14.

Figure 4-23 on page 4-37 shows the CLIDR bit assignments.

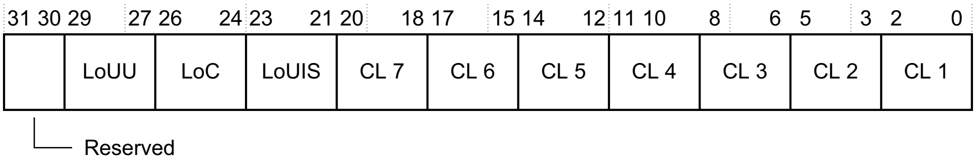


Figure 4-23 CLIDR bit assignments

Table 4-40 shows the CLIDR bit assignments.

Table 4-40 CLIDR bit assignments

Bits	Name	Description
[31:30]	Reserved	RAZ
[29:27]	LoUU	b001 = Level of Unification Uniprocessor
[26:24]	LoC	b001 = Level of Coherency
[23:21]	LoUIS	b001 = Level of Unification Inner Shared
[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 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
```

4.3.23 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 4-9 on page 4-14.

Table 4-41 shows the AIDR bit assignments.

Table 4-41 AIDR bit assignments

Bits	Name	Description
[31:0]	Reserved	RAZ

To access the Auxiliary Level ID Register, use:

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

Note

The AIDR is not used in this implementation.

4.3.24 Cache Size Selection Register

The CSSELR characteristics are:

Purpose Selects the cache described by the *Cache Size Identification Register* on page 4-35.

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 4-9 on page 4-14.

Figure 4-24 shows the CSSELR bit assignments.

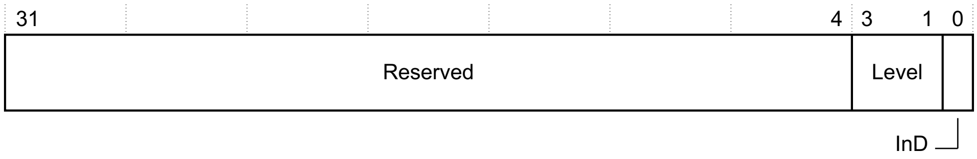


Figure 4-24 CSSELR bit assignments

Table 4-42 shows the CSSELR bit assignments.

Table 4-42 CSSELR bit assignments

Bits	Name	Description
[31:4]	Reserved	UNP or SBZ
[3:1]	Level	Cache level selected RAZ/WI There is only one level of cache in the Cortex-A5 processor so the value for this field is b000.
[0]	InD	`0x1` = Instruction cache `0x0` = 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
```

4.3.25 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
- banked, with some bits common to the Secure and Non-secure copies of the register.

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 modes when **CP15SDISABLE** is HIGH result in an Undefined instruction exception, see *Security Extensions write access disable* on page 3-10.

Attempts to modify read-only bits are ignored.

Configurations Available in all configurations.

Attributes See the register summary in Table 4-10 on page 4-15.

Figure 4-25 shows the SCTLR bit assignments.

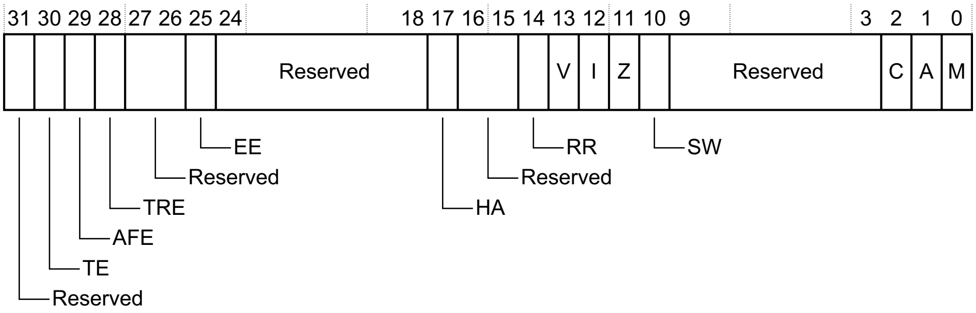


Figure 4-25 SCTLR bit assignments

Table 4-43 shows the SCTLR bit assignments.

Table 4-43 SCTLR bit assignments

Bits	Name	Access	Description
[31]	Reserved	-	RAZ/WI.
[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-A5 processor redefines the AP[0] bit as an access flag. You must invalidate the TLB after changing the AFE bit.

Table 4-43 SCTLR bit assignments (continued)

Bits	Name	Access	Description
[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.
[27:26]	Reserved	-	RAZ/WI
[25]	EE bit	Banked	Determines the value the CPSR.E bit is set to on an exception: 0 = Set to 0, little-endian. sets the reset value. 1 = Set to 1, big-endian. This value also indicates the endianness of the translation table data for translation table look-ups.
[24]	Reserved	-	RAZ/WI.
[23:22]	Reserved	-	RAO/WI.
[21]	Reserved	-	RAZ/WI.
[20:19]	Reserved	-	RAZ/WI.
[18]	Reserved	-	RAO/WI.
[17]	HA	-	RAZ/WI. Hardware management access flag disabled. The Cortex-A5 processor does not support this feature.
[16]	Reserved	-	RAO/WI
[15]	Reserved	-	RAZ/WI
[14]	RR	-	RAZ/WI. Cache replacement strategy. The Cortex-A5 processor only supports a fixed random replacement strategy.
[13]	V	Banked	Vectors bit. This bit selects the base address of the exception vectors: 0 = Normal exception vectors, base address `0x00000000`. This base address can be remapped. 1 = High exception vectors, Hivecs, base address `0xFFFF0000`. This base address is never remapped. At reset the value for this bit is taken from **VINITHI**.
[12]	I bit	Banked	Determines if instructions can be cached at any available cache level: 0 = instruction caching disabled at all levels. This is the reset value. 1 = instruction caching enabled.
[11]	Z bit	Banked	RAO/WI. Program flow prediction control. Branch prediction is always enabled on the Cortex-A5 processor when the MMU is 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]	Reserved	-	RAZ/WI.
[6:3]	Reserved	-	RAO/WI.

Table 4-43 SCTLR bit assignments (continued)

Bits	Name	Access	Description
[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.

To access the SCTLR, use:

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

4.3.26 Auxiliary Control Register

The ACTLR characteristics are:

Purpose Controls extended processor functionality from software, such as:

- behavior of the direct and indirect branch prediction in the *Prefetch Unit* (PFU)
- the limited dual issue capability of the *Data Processing Unit* (DPU).

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.

Attempts to write to this register in secure privileged modes when **CP15SDISABLE** is HIGH result in an Undefined instruction exception, see *Security Extensions write access disable* on page 3-10.

Configurations Available in all configurations.

Attributes See the register summary in Table 4-10 on page 4-15.

Figure 4-26 on page 4-42 shows the ACTLR bit assignments.

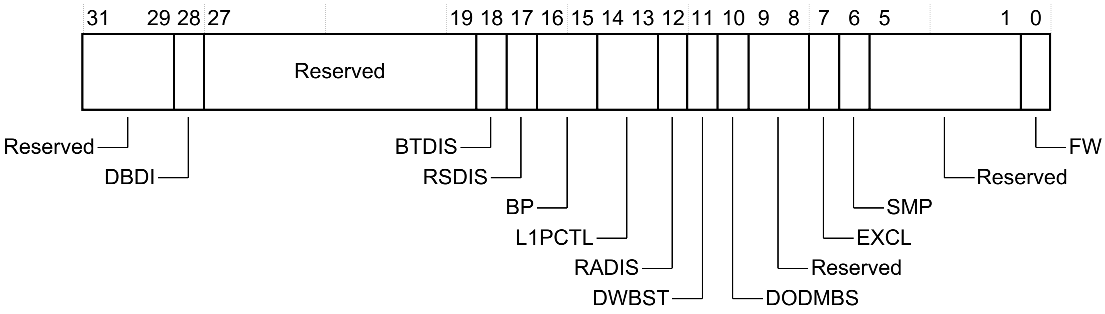


Figure 4-26 ACTLR bit assignments

Table 4-44 shows the ACTLR bit assignments.

Table 4-44 ACTLR bit assignments

Bits	Name	Description
[31:29]	Reserved	RAZ/WI.
[28]	DBDI	Disable branch dual issue.
[27:19]	Reserved	RAZ/WI.
[18]	BTDIS	Disable indirect *Branch Target Address Cache* (BTAC).
[17]	RSDIS	Disable return stack operation.
[16:15]	BP	Branch prediction policy: 00 = Normal operation 01 = Branch always taken 10 = Branch always not taken 11 = Reserved (Unpredictable).
[14:13]	L1PCTL	L1 Data prefetch control. The value of this field determines the maximum number of outstanding data prefetches allowed in the L1 memory system, not counting those generated by software load/PLD instructions: 00 = prefetch disabled 01 = 1 outstanding prefetch allowed 10 = 2 outstanding prefetches allowed 11 = 3 outstanding prefetches allowed.
[12]	RADIS	Disable Data Cache read-allocate mode. See *Read allocate mode* on page 2-4.
[11]	DWBST	Disable AXI data write bursts to Normal memory. Burst writes of length greater than one are output as single writes. The RADIS bit must be set to 1 when DWBST is set to 1. This bit does not change the behavior of cache evictions.
[10]	DODMBS	Disable optimized Data Memory Barrier behavior.
[9:8]	Reserved	RAZ/WI.
[7]	EXCL[a]	Exclusive L1/L2 cache control. The exclusive cache configuration does not permit data to reside in the L1 and L2 caches 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. For this feature to operate correctly, the L2 cache controller must also be configured for exclusive caching.

Table 4-44 ACTLR bit assignments (continued)

Bits	Name	Description
[6]	SMP	If this bit is set, data requests with Inner Cacheable Shared attributes are treated as cacheable. 0 = disabled. 1 = enabled.
[5:1]	Reserved	RAZ/WI.
[0]	FW	RAZ/WI.

a. This feature must only be enabled when the Cortex-A5 processor AXI master interface is directly connected to a PL310 L2 cache controller.

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
```

4.3.27 Coprocessor Access Control Register

The CPACR characteristics are:

Purpose Sets access rights for coprocessors CP10 and CP11, that support the VFP and Advanced SIMD extensions, if implemented..

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.

Behavior is unpredictable if the value of the CP11 field is not the same as the value of the CP10 field.

Configurations Available in all configurations.

Attributes See the register summary in Table 4-10 on page 4-15.

Figure 4-27 shows the CPACR bit assignments.

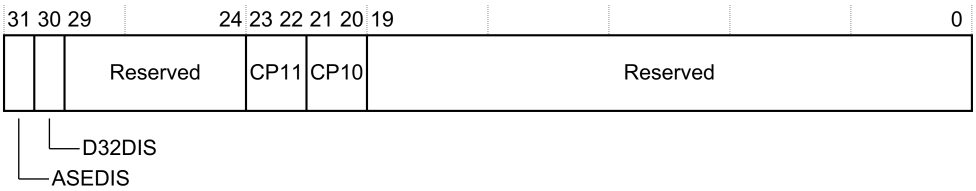


Figure 4-27 CPACR bit assignments

Table 4-45 shows the CPACR bit assignments.

Table 4-45 CPACR bit assignments

Bits	Name	Description
[31]	ASEDIS	Disable Advanced SIMD Extension functionality: 0 = All Advanced SIMD and VFP instructions execute normally. 1 = All Advanced SIMD instructions that are not VFP instructions are Undefined. See the *Cortex-A5 Floating-Point Unit Technical Reference Manual* and *Cortex-A5 NEON Media Processing Engine Technical Reference Manual* for more information. If implemented with floating point only, no NEON, this bit is RAO/WI. If implemented without both floating point and NEON, this bit is UNK/SBZP.
[30]	D32DIS	Disable use of registers D16-D31 of the VFP register file: 0 = All instructions accessing D0-D31 execute normally. 1 = Any VFP instruction that attempts to access any of registers D16-D31 is undefined. See the *Cortex-A5 Floating-Point Unit Technical Reference Manual* and *Cortex-A5 NEON Media Processing Engine Technical Reference Manual* for more information. If implemented with floating point only, no NEON, this bit is RAO/WI. If implemented without both floating point and NEON, this bit is UNK/SBZP.
[29:24]	Reserved	RAZ/WI.
[23:22]	CP11	Defines access permissions for the coprocessor. Access denied is the reset condition and is the behavior for nonexistent coprocessors. 00 = Access denied. This is the reset value. Attempted access generates an Undefined instruction exception. 01 = Privileged mode access only. 10 = Reserved. 11 = 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. 00 = Access denied. This is the reset value. Attempted access generates an Undefined instruction exception. 01 = Privileged mode access only. 10 = Reserved. 11 = Privileged and User mode access.
[19:0]	Reserved	RAZ/WI.

The *ARM Architecture Reference Manual* defines the VFP and Advanced SIMD instruction sets.

Access to coprocessors in the Non-secure state depends on the permissions set in the *Non-secure Access Control Register* on page 4-48.

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 4-48. Table 4-46 shows the results of attempted access to coprocessor access bits for each mode.

Table 4-46 Results of access to the CPACR

NSACR[11:10]	Secure privileged	Non-secure privileged	Secure or Non-secure User
b00	RW	RAZ/WI	Access prohibited[a]
b01	RW	RW	Access prohibited[a]

[a] User privilege access generates an Undefined Instruction exception.

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 will remain set to b00.

Note

You must enable both CP10 and CP11 before accessing any NEON or VFP system registers.

4.3.28 Secure Configuration Register

The SCR characteristics are:

Purpose Contains the fields used by the TrustZone secure monitor to control the overall Secure or Non-secure state of the Cortex-A5 processor. They can be used to access the banked CP15 registers and also to alter the behavior of the CPU when external asynchronous events are detected.

Usage constraints The SCR is:

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

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

Configurations Available in all configurations.

Attributes See the register summary in Table 4-10 on page 4-15.

Figure 4-28 shows the SCR bit assignments.

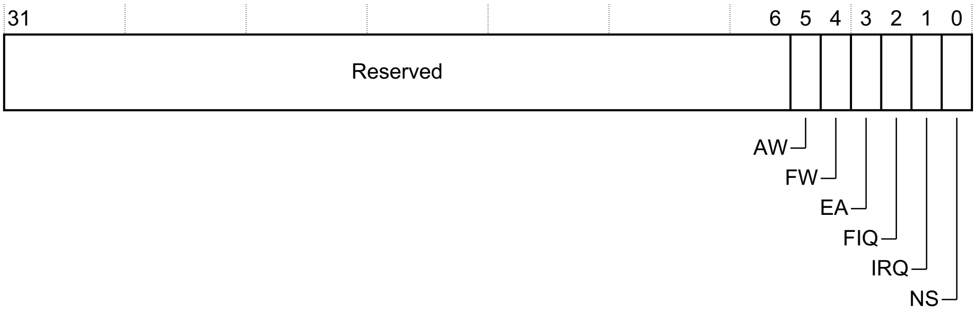


Figure 4-28 SCR bit assignments

Table 4-47 shows the SCR bit assignments.

Table 4-47 SCR bit assignments

Bits	Name	Description
[31:6]	Reserved	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. See Table 4-49 on page 4-47 for more information.
[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. See Table 4-48 for more information.
[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. See Table 4-49 on page 4-47 for more information.
[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. See Table 4-48 for more information.
[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	In modes other than Monitor mode, indicates whether the processor is in Secure or Non-secure state. In Monitor mode, indicates whether CP15 register accesses are to the Secure or the Non-secure view of the registers.: 0 = Secure. This is the reset value. 1 = Non-secure.

The values of the bits in the SCR have security implications. Table 4-48 shows the results for combinations of the FW and FIQ bits.

Table 4-48 Operation of the SCR 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. Do not use this combination of values.
0	0	The core might enter an infinite loop for Non-secure FIQ. Do not use this combination of values.

Table 4-49 shows the results for combinations of the AW and EA bits.

Table 4-49 Operation of the SCR AW and EA bits

AW	EA	Description
1	0	External aborts handled locally.
0	1	All external aborts trapped to Monitor mode.
1	1	All external aborts trapped to Monitor mode but the Non-secure state can hide secure aborts from the Monitor. Do not use this combination of values.
0	0	The core can unexpectedly enter an abort mode in the Non-secure state. Do not use this combination of values.

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
```

4.3.29 Secure Debug Enable Register

The SDER characteristics are:

Purpose Controls processor 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 4-10 on page 4-15.

Figure 4-29 shows the SDER bit assignments.

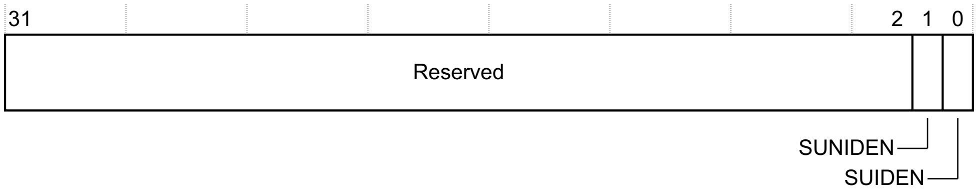


Figure 4-29 SDER bit assignments

Table 4-50 shows the SDER bit assignments.

Table 4-50 SDER bit assignments

Bits	Name	Description
[31:2]	Reserved	RAZ.
[1]	SUNIDEN	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]	SUIDEN	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
```

4.3.30 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 4-10 on page 4-15.

Figure 4-30 shows the NSACR bit assignments.

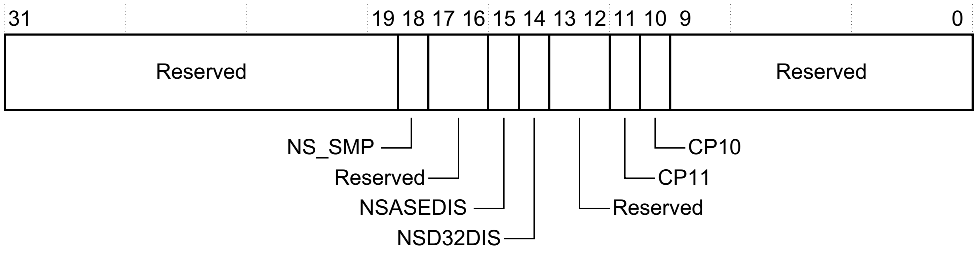


Figure 4-30 NSACR bit assignments

Table 4-51 shows the NSACR bit assignments.

Table 4-51 NSACR bit assignments

Bits	Name	Description
[31:19]	Reserved	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 instruction exception and the SMP bit is write ignored. This is the reset value. 1 = A write to Auxiliary Control Register in Non-secure state can modify the value of the SMP bit. Other bits are write ignored.
[17:16]	Reserved	RAZ/WI
[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.
[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.
[13:12]	Reserved	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]	Reserved	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
```

Table 4-52 shows the results of attempted access for each mode.

Table 4-52 Results of access to the NSACR

Secure privileged		Non-secure privileged		User	
Read	Write	Read	Write	Read	Write
Data	Data	Data	Undefined Instruction exception	Undefined Instruction exception	Undefined Instruction exception

See the *Cortex-A5 Floating-Point Unit Technical Reference Manual* and *Cortex-A5 NEON Media Processing Engine Technical Reference Manual* for more information.

4.3.31 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 4-10 on page 4-15.

Figure 4-31 shows the VCR bit assignments.

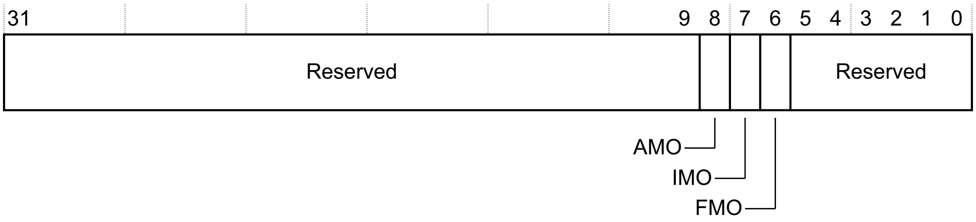


Figure 4-31 VCR bit assignments

Table 4-53 shows the VCR bit assignments.

Table 4-53 VCR bit assignments

Bits	Name	Description
[31:9]	Reserved	SBZ.
[8]	AMO	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 4-45.
[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 4-45.
[6]	FMO	FIQ Mask Override. When the processor is in Non-secure state and the SCR.FIQ bit is set, if the FMO 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 FMO bit is ignored. See *Secure Configuration Register* on page 4-45.
[5:0]	Reserved	SBZ.

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
```

4.3.32 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 modes when **CP15SDISABLE** is HIGH result in an Undefined instruction exception. See *Security Extensions write access disable* on page 3-10.

Configurations Available in all configurations.

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

Figure 4-32 shows the TTBR0 bit assignments. For an explanation of N in the figure, see *Translation Table Base Control Register* on page 4-53.

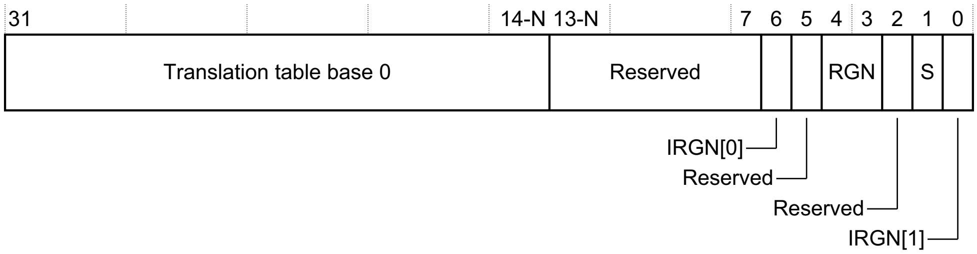


Figure 4-32 TTBR0 bit assignments

Table 4-54 shows the TTBR0 bit assignments. For an explanation of N in the table see *Translation Table Base Control Register* on page 4-53.

Table 4-54 TTBR0 bit assignments

Bits	Name	Description
[31:14-N]	Translation table base 0	Pointer to the level one translation table.
[13-N:7]	Reserved	UNP or SBZ.
[6]	IRGN[0]	Used with bit 0, IRGN[1] to describe inner cacheability.
[5]	Reserved	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 4-54 TTBR0 bit assignments (continued)

Bits	Name	Description
[2]	Reserved	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 = 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 non-cacheable.

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 4-53.

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
```

4.3.33 Translation Table Base Register 1

The TTBR1 characteristics are:

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

Usage constraints The 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 4-11 on page 4-15.

Figure 4-33 shows the TTBR1 bit assignments.

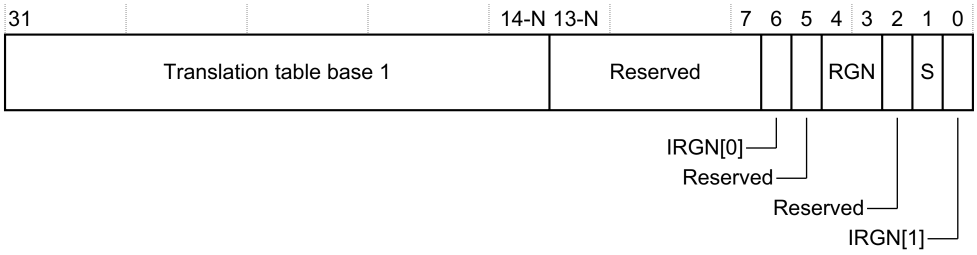


Figure 4-33 TTBR1 bit assignments

Table 4-55 shows the TTBR1 bit assignments.

Table 4-55 TTBR1 bit assignments

Bits	Name	Description
[31:14]	Translation table base 1	Pointer to the level one translation table.
[13:7]	Reserved	UNP or SBZ
[6]	IRGN[0]	Used with IRGN[1] to describe inner cacheability.
[5]	Reserved	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.
[2]	Reserved	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 non-cacheable.

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.

4.3.34 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 modes when **CP15SDISABLE** is HIGH result in an Undefined instruction exception. See *Security Extensions write access disable* on page 3-10.

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 4-11 on page 4-15.

Figure 4-34 shows the TTBCR bit assignments.

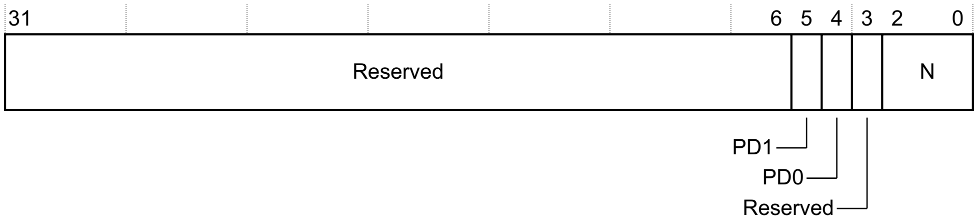


Figure 4-34 TTBCR bit assignments

Table 4-56 shows the TTBCR bit assignments.

Table 4-56 TTBCR bit assignments

Bits	Name	Description
[31:6]	Reserved	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 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 TTBR1 in use, the processor returns a section translation fault.
[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 TTBR0 in use, the processor returns a section translation fault.
[3]	Reserved	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.

4.3.35 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 modes when **CP15SDISABLE** is HIGH result in an Undefined instruction exception. See *Security Extensions write access disable* on page 3-10.

Configurations Available in all configurations.

Attributes See the register summary in Table 4-12 on page 4-15.

Figure 4-35 shows the DACR bit assignments.

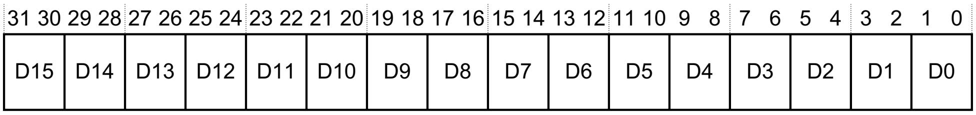


Figure 4-35 DACR bit assignments

Table 4-57 shows the DACR bit assignments.

Table 4-57 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
```

4.3.36 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 4-13 on page 4-16.

Figure 4-36 shows the DFSR bit assignments.

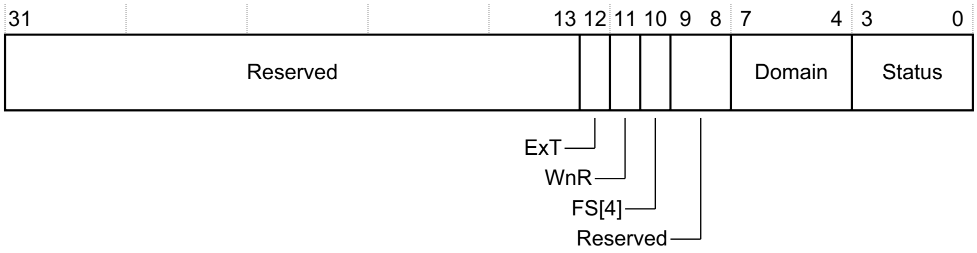


Figure 4-36 DFSR bit assignments

Table 4-58 shows the DFSR bit assignments.

Table 4-58 DFSR bit assignments

Bits	Name	Description
[31:13]	Reserved	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 bits [3:0] in this table.

Table 4-58 DFSR bit assignments (continued)

Bits	Name	Description
[9:8]	Reserved	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.

4.3.37 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 4-13 on page 4-16.

Figure 4-37 shows the IFSR bit assignments.

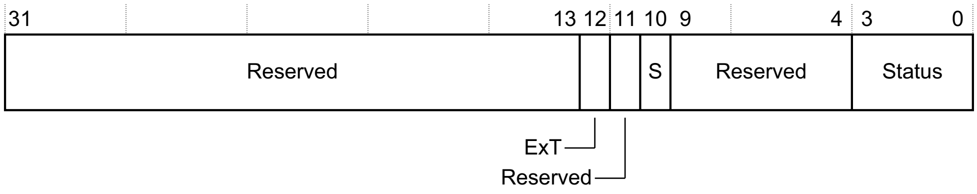


Figure 4-37 IFSR bit assignments

Table 4-59 shows the IFSR bit assignments.

Table 4-59 IFSR bit assignments

Bits	Name	Description
[31:13]	Reserved	UNP or SBZ.
[12]	ExT	External abort qualifier 0 = External abort marked as DECERR 1 = External abort marked as SLVERR.
[11]	Reserved	Always reads as 0.
[10]	S	Part of the status field. Must be used in conjunction with bit [12] and bits [3:0]
[9:4]	Reserved	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 4-45.

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.

4.3.38 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 4-13 on page 4-16.

Note

The ADFSR is not used 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
```

4.3.39 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 4-13 on page 4-16.

Note

The AIFSR is not used 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 the registers as RAZ. All writes are ignored.

4.3.40 Data Fault Address Register

The DFAR characteristics are:

Purpose Holds the MVA of 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 4-14 on page 4-16.

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.

4.3.41 Instruction Fault Address Register

The IFAR characteristics are:

Purpose Holds the MVA of the faulting 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 4-14 on page 4-16.

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 9-6.

4.3.42 NOP Register

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

4.3.43 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 4-15 on page 4-16.

The PAR 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 4-7 for information on Virtual Addresses.

The PAR 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 4-38. See *Data Fault Status Register* on page 4-56 for information on the Fault Status Register bits and the ExT bit. Figure 4-38 shows the bit assignment for PAR aborted translations.

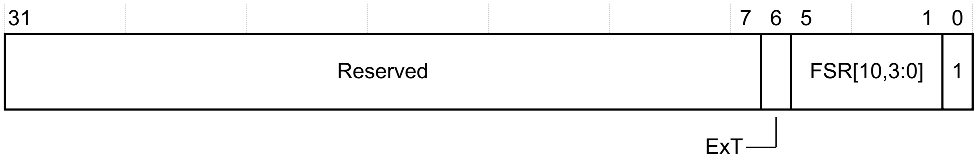


Figure 4-38 PAR aborted translation bit assignments

Figure 4-39 shows the PAR bit assignment if the translation completed successfully.

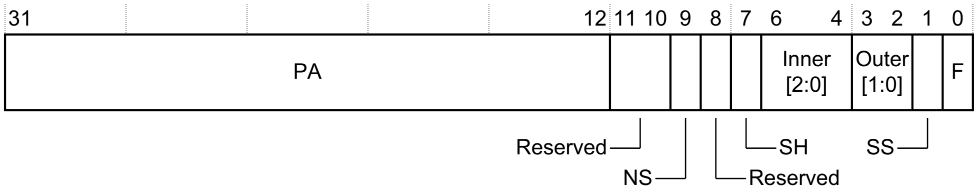


Figure 4-39 PAR successful translation bit assignments

Table 4-60 shows the PAR bit assignments.

Table 4-60 PAR bit assignments

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

Table 4-60 PAR bit assignments (continued)

Bits	Name	Description

[6:4]	Inner	Signals region inner attributes:
		b000 = Inner non-cacheable.
		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	Result of conversion: 0 = Successful translation.

4.3.44 Instruction Synchronization Barrier

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

4.3.45 Data Synchronization Barrier

The use of DSB is deprecated and, on Cortex-A5, behaves as NOP. Use the instruction `DSB` instead.

4.3.46 Data Memory Barrier

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

4.3.47 Vector Base Address Register

The VBAR characteristics are:

Purpose Provides the exception base address for exceptions that are not handled in monitor mode.

Usage constraints The VBAR is:

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

Configurations Available in all configurations.

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

Figure 4-40 on page 4-63 shows the VBAR bit assignments.

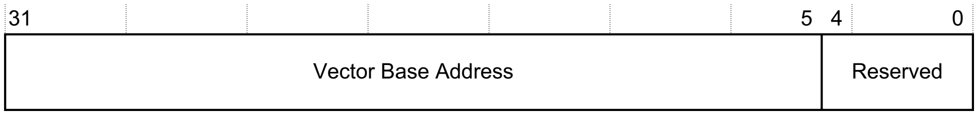


Figure 4-40 VBAR bit assignments

Table 4-61 shows the VBAR bit assignments.

Table 4-61 VBAR bit assignments

Bits	Name	Description
[31:5]	Vector Base Address	The base address of the normal exception vectors.
[4:0]	Reserved	UNK/SBZP.

To access the VBAR, use:

```
MRC p15, 0, <Rd>, c12, c0, 0 ; Read VBAR Register
MCR p15, 0, <Rd>, c12, c0, 0 ; Write VBAR Register
```

The Secure copy of the VBAR holds the vector base address for the Secure state, described as the Secure exception base address.

The Non-secure copy of the VBAR holds the vector base address for the Non-secure state, described as the Non-secure exception base address.

4.3.48 Monitor Vector Base Address Register

The MVBAR characteristics are:

Purpose Provides the exception base address for exceptions that are not handled in monitor mode.

Usage constraints The MVBAR is:

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

Configurations Available in all configurations.

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

Figure 4-41 shows the MVBAR bit assignments.

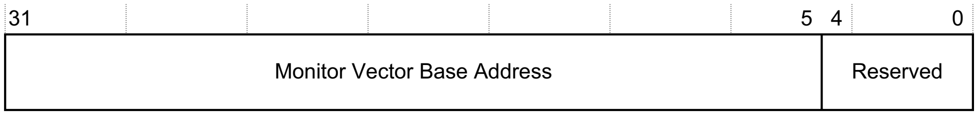


Figure 4-41 MVBAR bit assignments

Table 4-62 shows the MVBAR bit assignments.

Table 4-62 MVBAR bit assignments

Bits	Name	Description
[31:5]	Monitor Vector Base Address	The base address of the exception vectors for exceptions that are handled in Monitor mode.
[4:0]	Reserved	UNK/SBZP.

To access the MVBAR, use:

```
MRC p15, 0, <Rd>, c12, c0, 1 ; Read MVBAR Register
MCR p15, 0, <Rd>, c12, c0, 1 ; Write MVBAR Register
```

4.3.49 Interrupt Status Register

The ISR characteristics are:

Purpose Reports the status of virtual or physical interrupts.

Usage constraints The ISR 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 4-19 on page 4-19.

The ISR reports pending virtual interrupts and aborts in Non-secure state if the relevant mask override field in the VCR and exception taken in monitor mode bits in the SCR are set. Otherwise it reflects pending physical interrupts and aborts. The register continues to report a virtual interrupt until the corresponding field in the VIR is cleared.

Figure 4-42 shows the ISR bit assignments.

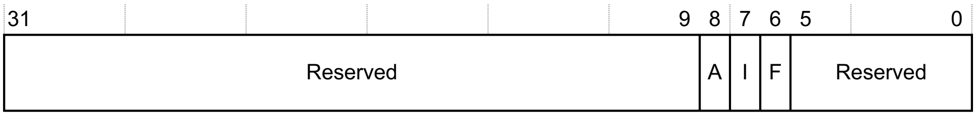


Figure 4-42 ISR bit assignments

Table 4-63 shows the ISR bit assignments.

Table 4-63 ISR bit assignments

Bits	Name	Description
[31:9]	Reserved	RAZ/UNK.
[8]	A	External abort pending flag: 0 = no pending external abort 1 = an external abort is pending.
[7]	I	Interrupt pending flag. Indicates whether an IRQ interrupt is pending: 0 = no pending IRQ 1 = an IRQ interrupt is pending.
[6]	F	Fast interrupt pending flag. Indicates whether an FIQ fast interrupt is pending: 0 = no pending FIQ 1 = an FIQ fast interrupt is pending.
[5:0]	Reserved	UNK/SBZP.

To access the ISR, use:

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

4.3.50 Virtualization Interrupt Register

The VIR characteristics are:

Purpose Triggers a pending interrupt from software running on the processor.

Usage constraints The VIR 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 4-19 on page 4-19.

Figure 4-43 shows the VIR bit assignments.

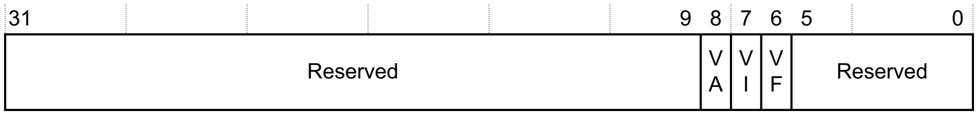


Figure 4-43 VIR bit assignments

Table 4-64 shows the VIR bit assignments.

Table 4-64 VIR bit assignments

Bits	Name	Description
[31:9]	Reserved	RAZ/WI.
[8]	VA	Virtual Abort bit. When set the virtual abort becomes pending when SCR.EA and VCR.AMO are b1. The virtual abort exception happens only when the processor is in Non-secure state and the CPSR.A field is clear.
[7]	VI	Virtual IRQ bit. When set the virtual interrupt becomes pending when SCR.IRQ and VCR.IMO are b1. The virtual interrupt exception happens only when the processor is in Non-secure state and the CPSR.I field is clear.
[6]	VF	Virtual FIQ bit. When set the virtual interrupt becomes pending when SCR.FIQ and VCR.FMO are b1. The virtual interrupt exception happens only when the processor is in Non-secure state and the CPSR.F field is clear.
[5:0]	Reserved	RAZ/WI.

To access the VIR, use:

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

A virtual abort or interrupt becomes pending when the associated abort/interrupt field in the SCR {EA, IRQ, FIQ} is set and the appropriate VCR mask override field in the VCR {AMO, IMO, FMO} is also set. The actual virtual interrupt/abort exception is taken only when the CPU enters Non-secure state and can be masked using the CPSR {A,I,F} fields.

When a virtual abort is taken, the corresponding bit in the VIR is cleared automatically by hardware. When a virtual interrupt is taken, the corresponding bit in the VIR is not changed by hardware. For the purposes of WFE and WFI, virtual interrupts and aborts are treated in the same way as normal interrupts and aborts.

4.3.51 Context ID Register

The CONTEXTIDR characteristics are:

Purpose Identifies the current *Process Identifier* (PROCID) and *Address Space Identifier* (ASID). The value of the whole of this register is called the Context ID and is used by the debug logic and the trace logic.

Usage constraints The CONTEXTIDR 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 4-20 on page 4-19.

Figure 4-44 shows the CONTEXTIDR bit assignments.

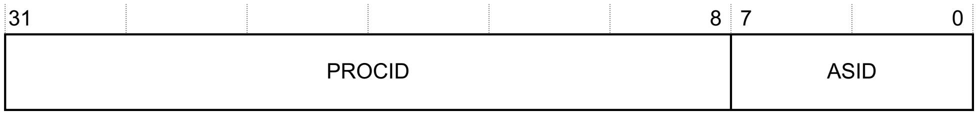


Figure 4-44 CONTEXTIDR bit assignments

Table 4-65 shows the CONTEXTIDR bit assignments.

Table 4-65 CONTEXTIDR bit assignments

Bits	Name	Description
[31:8]	PROCID	Process Identifier. This field must be programmed with a unique value that identifies the current process. It is used by the trace logic and the debug logic to identify the process that is currently running.
[7:0]	ASID	Address Space Identifier. This field is programmed with the value of the current ASID.

To access the CONTEXTIDR, use:

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

4.3.52 Configuration Base Address Register

The CBAR characteristics are:

Purpose This register is not implemented on Cortex-A5 and always reads as `0x00000000`.

Usage constraints The CBAR 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 4-21 on page 4-20.

To access the CBAR, use:

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

Chapter 5
Non-debug Use of CP14

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

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

5.1 About coprocessor CP14

Coprocessor CP14 provides support for the hardware acceleration of Java bytecodes.

5.2 CP14 Jazelle register summary

In the optional Cortex-A5 implementation of the Jazelle Extension:

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

Table 5-1 shows the CP14 Jazelle registers. All Jazelle registers are 32 bits wide.

Table 5-1 CP14 Jazelle registers summary

CRn	Op1	CRm	Op2	Name	Type	Reset	Description
0	7	0	0	JIDR	RW[a]	`0xF410014A`	*Jazelle Identity and Miscellaneous Functions Register* on page 5-4
7	1	0	0	JOSCR	RW	-	*Jazelle Operating System Control Register* on page 5-5
7	2	0	0	JMCR	RW	-	*Jazelle Main Configuration Register* on page 5-6
7	3	0	0	JPR	RW	-	*Jazelle Parameters Register* on page 5-7
7	4	0	0	JCOTTR	WO	-	*Jazelle Configurable Opcode Translation Table Register* on page 5-8

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

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

5.3 CP14 Jazelle register descriptions

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

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

5.3.1 Jazelle 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 5-5.

Configurations Available in all configurations.

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

Figure 5-1 shows the JIDR bit assignments.

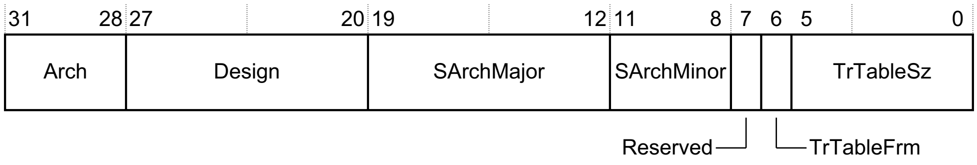


Figure 5-1 JIDR bit assignment

Table 5-2 shows the JIDR bit assignments.

Table 5-2 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]	Reserved	RAZ.
[6]	TrTbleFrm	Indicates the format of the *Jazelle Configurable Opcode Translation Table Register* (JCOTTR).
[5:0]	TrTbleSz	Indicates the size of the JCOTTR.

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 5-8.

5.3.2 Jazelle Operating System Control Register

The JOSCR characteristics are:

Purpose Enables operating systems to control access to Jazelle Extension hardware.

Usage constraints The JOSCR is:

- only accessible in privileged modes
- set to zero after a reset and must be written in privileged modes.

Configurations Available in all configurations.

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

Figure 5-2 shows the JOSCR bit assignments.

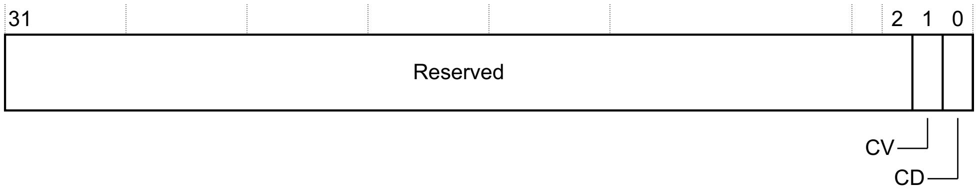


Figure 5-2 JOSCR bit assignments

Table 5-3 shows the JOSCR bit assignments.

Table 5-3 JOSCR bit assignments

Bits	Name	Description
[31:2]	Reserved	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
```

5.3.3 Jazelle Main Configuration Register

The JMCR characteristics are:

Purpose Describes the Jazelle hardware configuration and its behavior.

Usage constraints Only accessible in privileged modes.

Configurations Available in all configurations.

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

Figure 5-3 shows the JMCR bit assignments.

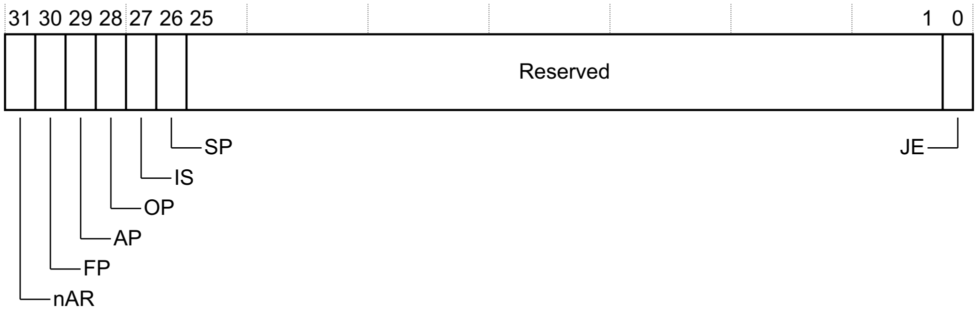


Figure 5-3 JMCR bit assignments

Table 5-4 shows the JMCR bit assignments.

Table 5-4 JMCR bit assignments

Bits	Name	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.

Table 5-4 JMCR bit assignments (continued)

Bits	Name	Description
[27]	IS	The *Index Size* (IS) bit specifies the size of the index associated with quick object field accesses: 0 = Quick object field indices are 8 bits. 1 = Quick object field indices are 16 bits.
[26]	SP	The *Static Pointer* (SP) bit controls how the Jazelle hardware treats static references: 0 = Static references are treated as handles. 1 = Static references are treated as pointers.
[25:1]	Reserved	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
```

5.3.4 Jazelle Parameters Register

The JPR 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 5-1 on page 5-3.

Figure 5-4 shows the JPR bit assignments.

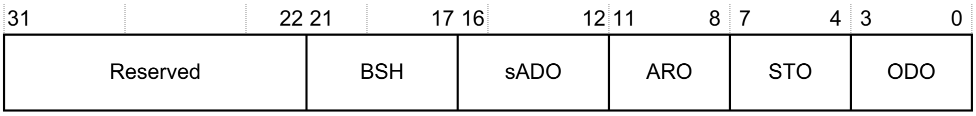


Figure 5-4 JPR bit assignments

Table 5-5 shows the JPR bit assignments.

Table 5-5 Jazelle Parameters Register bit assignments

Bits	Name	Description
[31:22]	Reserved	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 JPR, use:

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

5.3.5 Jazelle Configurable Opcode Translation Table Register

The JCOTTR characteristics are:

Purpose Provides translations between the configurable opcodes in the range `0xCB`-`0xFD` and the operations that are provided by the Jazelle hardware.

Usage constraints Only accessible in privileged modes.

Configurations Available in all configurations.

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

Figure 5-5 shows the JCOTTR bit assignments.

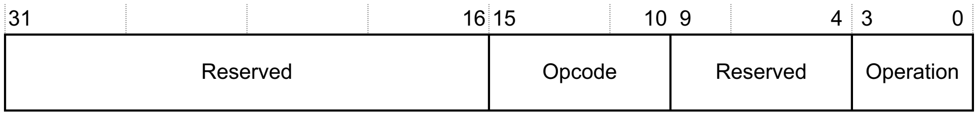


Figure 5-5 JCOTTR bit assignments

Table 5-6 shows the JCOTTR bit assignments.

Table 5-6 JCOTTR bit assignments

Bits	Name	Description
[31:16]	Reserved	SBZ

Table 5-6 JCOTTR bit assignments (continued)

Bits	Name	Description
[15:10]	Opcode	Contains the bottom bits of the configurable opcode.
[9:4]	Reserved	SBZ
[3:0]	Operation	Contains the code for the operation `0x0`- `0x9`

To access this register, use:

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

Chapter 6
Memory Management Unit

This chapter describes the *Memory Management Unit* (MMU). It contains the following sections:

6.1 About the MMU

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

The ARM v7 *Virtual Memory System Architecture* (VMSA) features include the following:

- Page table entries that support:
  - 16MB supersections. The processor supports supersections that consist of 16MB blocks of memory.
  - 1MB sections.
  - 64KB large pages.
  - 4KB small pages.
- 16 access domains
- Global and application-specific identifiers to remove the requirement for context switch TLB flushes.
- Extended permissions checking capability.

TLB maintenance and configuration operations are controlled through a dedicated coprocessor, CP15, integrated with the core. This coprocessor provides a standard mechanism for configuring the L1 memory system.

See the *ARM Architecture Reference Manual* for a full architectural description of the ARMv7 VMSA.

6.2 Memory management system

The Cortex-A5 processor supports the ARM v7 VMSA including the TrustZone security extension. The translation of a *Virtual Address* (VA) used by the instruction set architecture to a *Physical Address* (PA) used in the memory system and the management of the associated attributes and permissions is carried out using a two-level MMU.

The first level MMU uses a Harvard design with separate micro TLB structures in the PFU for instruction fetches (IuTLB) and in the DPU for data read and write requests (DuTLB).

A miss in the micro TLB results in a request to the main unified TLB shared between the data and instruction sides of the memory system. The TLB consists of a 128-entry two-way set-associative RAM based structure. The TLB page-walk mechanism supports page descriptors held in the L1 data cache. The caching of page descriptors is configured globally for each translation table base register, TTBRx, in the system coprocessor, CP15.

The TLB contains a hitmap cache of the page types which have already been stored in the TLB. When the processor is put into dormant mode as described in *Power control* on page 2-9, the TLB RAM data is retained. To save and restore the hitmap, the register can be accessed directly from software using CP15 as described in *c15, TLB access and attributes* on page 4-12.

6.2.1 Memory types

Although various different memory types can be specified in the page tables, the Cortex-A5 processor does not implement all possible combinations:

- Write-through caches are not supported. Any memory marked as write-through is treated as Non-cacheable.
- The outer shareable attribute is not supported. Anything marked as outer shareable is treated in the same way as inner shareable.
- Write-back no write allocate is not supported. It is treated as write-back write-allocate.

Table 6-1 shows the treatment of each different memory type in the Cortex-A5 processor in addition to the architectural requirements.

Table 6-1 Treatment of memory attributes

Memory type attribute	Shareability	Other attributes	Notes
Strongly Ordered	-	-	-
Device	Non-shareable	-	-
	Shareable	-	-

Table 6-1 Treatment of memory attributes (continued)

Memory type attribute	Shareability	Other attributes	Notes
Normal	Non-shareable	Non-cacheable	Does not access L1 caches.
		Write-through cacheable	Treated as non-cacheable.
		Write-back cacheable, write allocate	Can dynamically switch to no write allocate, if more than three full cache lines are written in succession. See *Read allocate mode* on page 2-4.
		Write-back cacheable, no write allocate	Treated as non-shareable write-back cacheable, write allocate
	Inner shareable	Non-cacheable	-
		Write-through cacheable	Treated as inner shareable non-cacheable.
		Write-back cacheable, write allocate	Treated as inner shareable non-cacheable unless the SMP bit in the Auxiliary Control Register is set (ACTLR[6] = b1). See *Auxiliary Control Register* on page 4-41. If this bit is set the area is treated as Write-back cacheable write allocate.
		Write-back cacheable, no write allocate	
	Outer shareable	Non-cacheable	Treated as inner shareable non-cacheable.
		Write-through cacheable	
		Write-back cacheable, write allocate	Treated as inner shareable non-cacheable unless the SMP bit in the Auxiliary Control Register is set (ACTLR[6] = b1). See *Auxiliary Control Register* on page 4-41. If this bit is set the area is treated as Write-back cacheable write allocate.
		Write-back cacheable, no write allocate	

6.3 TLB organization

TLB organization is described in the following sections:

- *Micro TLB*
- *Main TLB*.

6.3.1 Micro TLB

The first level of caching for the page table information is a micro TLB of 10 entries that is implemented on each of the instruction and data sides. These blocks provide a lookup of the virtual addresses in a single cycle.

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

All main TLB related maintenance operations affect both the instruction and data micro TLBs, causing them to be flushed. In the same way, any change of the following registers causes the micro TLBs to be flushed:

- *Context ID Register* (CONTEXTIDR)
- *Domain Access Control Register* (DACR)
- *Primary Region Remap Register* (PRRR)
- *Normal Memory Remap Register* (NMRR)
- *Translation Table Base Registers* (TTBR0 and TTBR1).

6.3.2 Main TLB

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.

The main TLB is 128-entry two-way set-associative.

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 (ASID), or as global for all application spaces. The CONTEXTIDR determines the currently selected application space.

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 in the CONTEXTIDR or is global.

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

6.4 Memory access sequence

When the processor generates a memory access, the MMU:

1. Performs a lookup for the requested virtual address and current ASID and security state in the relevant instruction or data micro TLB.
2. If there is a miss in the micro TLB, performs a lookup for the requested virtual address and current ASID and security state in the main TLB.
3. If there is a miss in 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 *Translation Table Base Register 0* on page 4-51 and *Translation Table Base Register 1* on page 4-52. If the encoding of the IRGN bits is write-back, an L1 data cache lookup is performed and data is read from the data cache. If the encoding of the IRGN bits is write-through or non-cacheable, 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 *Translation Table Base Control Register* on page 4-53. If translation table walks are disabled, the processor returns a Section Translation fault.

If the TLB finds a matching 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 *Instruction Fault Status Register* (IFSR) and *Data Fault Status Register* (DFSR).
2. The memory region attributes specified in both the TLB entry and the CP15 c10 remap registers 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 4-10.
3. The TLB translates the virtual address to a physical address for the memory access.

6.5 Interaction with memory system

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

6.6 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 Cortex-A5 processor. External aborts can be configured to trap to Monitor mode by setting the EA bit in the *Secure Configuration Register* on page 4-45.

6.6.1 External aborts on data write

Externally generated errors during a data 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.

Externally generated errors during data read are always synchronous. The address captured in the DFAR matches the address which generated the external abort.

6.6.2 Synchronous and asynchronous aborts

Chapter 4 *System Control* 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.

6.7 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 6-2.

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 6-2 CP15 register functions

Register	Cross reference
TLB Type Register	*TLB Type Register* on page 4-23.
Control Register	*System Control Register* on page 4-38
Non-secure Access Control Register	*Non-secure Access Control Register* on page 4-48
Translation Table Base Register 0	*Translation Table Base Register 0* on page 4-51
Translation Table Base Register 1	*Translation Table Base Register 1* on page 4-52
Translation Table Base Control Register	*Translation Table Base Control Register* on page 4-53
Domain Access Control Register	*Domain Access Control Register* on page 4-55
DFSR	*Data Fault Status Register* on page 4-56
IFSR	*Instruction Fault Status Register* on page 4-57
DFAR	*Data Fault Address Register* on page 4-60
IFAR	*Instruction Fault Address Register* on page 4-60
TLB operations	*c8, TLB maintenance operations* on page 4-9.
Primary Region Remap Register	*c10, Memory region remap* on page 4-10
Normal Memory Remap Register	
TLB Hitmap Register	*c15, TLB access and attributes* on page 4-12.
Context ID Register	*Context ID Register* on page 4-66

Chapter 7
Level 1 Memory System

This chapter describes the *Level 1* (L1) memory system. It contains the following sections:

7.1 About the L1 memory system

The L1 memory system has:

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

The data side of the L1 memory system has:

- two 16-byte linefill buffers and one 32-byte eviction buffer
- a 4-entry, 64-bit merging store buffer.

7.1.1 Memory system

This section describes:

- *Cache features*
- *Store buffer*.

Cache features

The Cortex-A5 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 4-38.
- Cache replacement policy is pseudo random.
- Data cache is 4-way set-associative.
- Instruction cache is 2-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 4KB, 8KB, 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.

Instruction cache features

The instruction cache is virtually indexed and physically tagged.

Data cache features

The data cache is physically indexed and physically tagged.

Store buffer

The Cortex-A5 processor has a store buffer with four 64-bit slots with data merging capability.

7.2 Security extensions support

The Cortex-A5 processor supports the TrustZone architecture, and exports the Secure or Non-secure status of its memory requests to the memory system.

7.3 L1 instruction side memory system

The L1 instruction side memory system is responsible for providing an instruction stream to the Cortex-A5 processor. To increase overall performance and to reduce power consumption, it contains the following functionality:

- dynamic branch prediction
- Instruction caching.

The ISide comprises the following:

Prefetch Unit (PFU)

The PFU implements a two-level prediction mechanism, comprising the following:

- A 256 entry branch pattern history table.
- A four-entry BTAC.
- A four-entry return stack.

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, or 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 two-way set-associative. It comprises the following features:

- configurable sizes of 4KB, 8KB, 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.

7.3.1 Enabling program flow prediction

Program flow prediction is always enabled when the MMU is enabled by setting the M-bit in the CP15 c1 Control register to 1. See *System Control Register* on page 4-38.

7.3.2 Program flow prediction

The following sections describe program flow prediction:

- *Predicted and non-predicted instructions* on page 7-5
- *Thumb state conditional branches* on page 7-5
- *Return stack predictions* on page 7-5.

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 associated with function-call and return instructions
- branches that switch between ARM and Thumb states.

However, some branch instructions are not predicted:

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

Thumb state conditional branches

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

Return stack predictions

The return stack stores the address and the 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` immediate
- `BLX` register.

The following instructions cause a return stack pop if predicted:

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

The `LDR` instruction can use any of the addressing modes, as long as r13 is the base register.

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

7.4 L1 data side memory system

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

7.4.1 Internal exclusive monitor

The Cortex-A5 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 *Watchpoint Control Register* on page 9-25.

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 direct 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.

However, if the address LDREX/STREX code sequence is in cacheable memory, any eviction of the cache line containing that address clears the monitor. It is therefore recommended that no load or store instructions are placed between the LDREX and STREX because these additional instructions can cause a cache eviction.

7.4.2 External aborts handling

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

- All load accesses use the synchronous abort mechanism.
- All store accesses use the asynchronous abort mechanism, except for SWP and SWPB to any type of memory, and STREX, STREXB, STREXH, and STREXD to shareable non-cacheable normal or shareable device or strongly-ordered memory.

7.5 Data prefetching

This section describes:

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

7.5.1 The PLD instruction

`PLD` instructions lookup in the cache, and start a linefill if they missed, just like a normal load instruction. However, the `PLD` instruction retires immediately rather than waiting for the data to arrive in the cache, which enables other instructions to execute while the linefill continues in the background.

7.5.2 Data prefetching and monitoring

The Cortex-A5 data cache implements an automatic prefetcher that monitors cache misses done by the processor. When a pattern is detected, the automatic prefetcher starts linefills in the background. Occasionally these linefills might be dropped before the data is allocated into the cache. It can be deactivated in software using a CP15 Auxiliary Control Register bit. See *Auxiliary Control Register* on page 4-41.

Note

Automatic data prefetching is only performed for memory that is marked as non-shared.

7.6 Direct access to internal memory

The Cortex-A5 processor provides a mechanism to read the internal memory used by the Cache and TLB structures through the implementation-defined region of the system coprocessor interface. This functionality can be useful when investigating issues where the coherency between the data in the cache and data in system memory is broken.

The appropriate memory block and location is selected using a number of write-only CP15 registers and the data is read from a pair of read-only CP15 registers as shown in Table 7-1. These operations are only available in secure privileged modes. In all other modes, executing the CP15 instruction results in an Undefined instruction exception.

Table 7-1 Cortex-A5 system coprocessor CP15 registers used to access internal memory

Function	Access	CP15 operation	Rd Data
Data Register 0	Read-only	`MRC p15, 3, <Rd>, c15, c0, 0`	Data
Data Register 1	Read-only	`MRC p15, 3, <Rd>, c15, c0, 1`	Data
Data Cache Tag Read Operation Register	Write-only	`MCR p15, 3, <Rd>, c15, c2, 0`	Set/Way
Instruction Cache Tag Read Operation Register	Write-only	`MCR p15, 3, <Rd>, c15, c2, 1`	Set/Way
Data Cache Data Read Operation Register	Write-only	`MCR p15, 3, <Rd>, c15, c4, 0`	Set/Way/Offset
Instruction Cache Data Read Operation Register	Write-only	`MCR p15, 3, <Rd>, c15, c4, 1`	Set/Way/Offset
TLB Data Read Operation Register	Write-only	`MCR p15, 3, <Rd>, c15, c4, 2`	Index/Way

The encodings for the operations and the format for the data read from the memory are described in the following sections:

- *Data Cache Tag and Data encoding*
- *Instruction Cache Tag and Data encoding* on page 7-9
- *TLB data encoding* on page 7-10.

7.6.1 Data Cache Tag and Data encoding

The Cortex-A5 processor data cache consists of a 4-way set-associative structure. The number of sets in each way depends on the configured size of the cache. The encoding, set in Rd in the appropriate `MCR` instruction, used to locate the required cache data entry for tag and data memory is shown in Table 7-2. It is very similar for both the tag and data RAM access. Data RAM access includes an additional field to locate the appropriate words in the cache line. The set-index range parameter (S) is determined by:

S = $\log_2$(Data cache size / 4).

Table 7-2 Data Cache Tag and Data location encoding

Bit-field of Rd	Description
[31:30]	Cache Way
[29:S+5]	Unused
[S+4:5]	Set index
[4:2]	Cache word data offset (Data Register only)
[1:0]	Unused

The tag information (dirty state, outer attributes, and valid) for the selected cache line is returned using only Data Register 0 using the format shown in Table 7-3.

Table 7-3 Data Cache Tag data format

Bit-field of Data Register 0	Description
[31:30]	Unused
[29]	Valid
[28]	TrustZone Non-secure state (NS)
[27:6]	Tag Address
[5:3]	Unused
[2:1]	Outer memory attributes
[0]	Dirty state

The 32 bits of cache data is also returned in Data register 0.

7.6.2 Instruction Cache Tag and Data encoding

The Cortex-A5 processor instruction cache is significantly different from the data cache and this is shown in the encodings and data format used in the CP15 operations used to access the Tag and data memories. Table 7-4 shows the encoding required to select a given cache line. The set-index range parameter (S) is determined by:

$S = \log_2$(Instruction cache size / 2) for the 2-way set associative cache.

Table 7-4 Instruction Cache Tag and Data location encoding

Bit-field of Rd	Description
[31]	Cache Way
[30:S+5]	Unused
[S+4:5]	Set index
[4:2]	Cache data element offset (Data Register only)
[1:0]	Unused

The Tag and valid bits for the selected cache line are returned using only Data Register 0 using the format shown in Table 7-5.

Table 7-5 Instruction Cache Tag data format

Bit-field of Data Register 0	Description
[31:24]	Unused.
[23:22]	Valid bits
[21]	TrustZone Non-secure state (NS).
[20:0]	Tag Address.

The CP15 Instruction Cache Data Read Operation returns two entries from the cache in Data Register 0 and Data Register 1 corresponding to the 16-bit aligned offset in the cache line:

Data Register 0 Data from cache offset+b00.

Data Register 1 Data from cache offset+b10.

In ARM mode these two fields combined always represent a single instruction. In Thumb they can represent any combination of 16-bit and partial or full 32-bit instructions.

7.6.3 TLB data encoding

The Cortex-A5 processor unified TLB is built from a two-way set-associative RAM based structure. The individual entries can be read into the data registers by writing to the TLB Data Read Operation Register using the encoding shown in Table 7-6.

The set-index range parameter (S) is determined by:

S = $\log_2$(No. TLB entries / 2).

For the Cortex-A5 processor, No. TLB entries is 128.

Table 7-6 TLB Data Read Operation Register location encoding

Bit-field of Rd	Description
[31]	TLB Way
[30:S]	Unused
[S-1:0]	TLB index

The TLB uses a 63-bit encoding for the descriptor which is returned in both Data Registers:

Data Register 0[31:0] TLB Descriptor[31:0].

Data Register 1[30:0] TLB Descriptor[62:32].

Table 7-7 shows the data fields in the TLB descriptor.

Table 7-7 TLB descriptor format

Bit-field	Field name	Width	Comments
[62:59]	Domain	4	-
[58:56]	Access Permissions (AP)	3	-
[55]	NS (Non-Secure Walk Flag)	1	The security state during the page-walk that fetched the entry.
[54]	NS (Non-Secure Page Flag)	1	The security state allocated to this memo
[53]	XN	1	Execute Never
[52:50]	TEX	3	-
[49]	Bufferable	1	-
[48]	Cacheable	1	-
[47]	Shareable	1	-
[46:39]	ASID	8	The current ContextID

Table 7-7 TLB descriptor format (continued)

Bit-field	Field name	Width	Comments

[38:37]	Size	2	b00 = Small Page
			b01 = Large Page
			b10 = Section
			b11 = SupersSection

[36:22]	VA	15	-
[21:2]	PA	20	-
[1]	Not-Global (nG)	1	b0 = entry is global to all ASIDs b1 = entry is local to listed ASID
[0]	Valid	1	-

Chapter 8
Level 2 Memory Interface

This chapter describes the *Level 2* (L2) memory interface. It contains the following sections:

- *About the L2 interface* on page 8-2
- *AXI privilege information* on page 8-6.

8.1 About the L2 interface

The L2 AXI interface enables the L1 memory system to have access to peripherals and to external memory using an AXI master port. The L2 interface is described in:

- *AXI master interface*
- *L2 memory interface attributes*
- *Supported AXI transfers* on page 8-3
- *AXI transaction IDs* on page 8-3
- *AXI user bits* on page 8-4
- *Write response* on page 8-4
- *Exclusive L2 cache* on page 8-5.

8.1.1 AXI master interface

The AXI master interface provides a high bandwidth interface to second level caches, on-chip RAM, peripherals, and interfaces to external memory. It consists of a single AXI port with a 64-bit read channel for instruction fetches and data reads and a 64-bit write channel for data writes.

The AXI master can run at the same frequency as the processor, or at a lower synchronous frequency. If asynchronous clocking is required an external asynchronous AXI bridge is required.

8.1.2 L2 memory interface attributes

Table 8-1 shows the AXI master interface attributes.

Table 8-1 AXI master interface attributes

Attribute	Value	Comments
Write issuing capability	64	Made up of: • 33 outstanding writes to normal memory, which can be evictions, single writes, or write bursts, of which a maximum of 18 can be to shareable cacheable memory, and a maximum of 15 to non-cacheable memory • up to 31 outstanding writes to device or strongly ordered memory, which can be single writes or write bursts.
Read issuing capability	8	Made up of five linefills on the data side, one non-cacheable data read, one non-cacheable TLB read, and one instruction fetch.
Write ID capability	3	Made up of one for cacheable writes, one for non-cacheable writes, and one for device and strongly ordered writes.
Write ID width	2	-
Write interleave capability	1	The AXI master interface presents all write data in order.
Read ID capability	5	Made up of one for instruction fetches, one for TLB reads, and three for data reads and linefills.
Read ID width	3	-
Combined issuing capability	72	-

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

8.1.3 Supported AXI transfers

The Cortex-A5 processor master port generates 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 or WRAP4 64-bit for write transfers (evictions)
- INCR N (N:1-4) 64-bit write transfers.

For non-cacheable transactions:

- INCR N (N:1-4) 64-bit read transfers
- INCR N (N:1-4) 64-bit write 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 exclusive read transfers
- INCR 1 8-bit, 16-bit, 32-bit, and 64-bit exclusive write transfers
- INCR 1 32-bit read/write (locked) for swap
- INCR 1 8-bit read/write (locked) for swap.

For device, or strongly ordered transactions only:

- INCR N (N:1-8) 32-bit read transfers
- INCR N (N:1-8) 32-bit write transfers
- INCR 1 8-bit, 16-bit, and 32-bit read transfers
- INCR 1 8-bit, 16-bit, and 32-bit write transfers
- INCR 1 8-bit, 16-bit, 32-bit, and 64-bit exclusive read transfers
- INCR 1 8-bit, 16-bit, 32-bit, and 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 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 none, some, or all byte strobes low can occur.

8.1.4 AXI transaction IDs

Table 8-2 shows the AXI ID signal encodings.

Table 8-2 AXI ID signal encodings

Request type	ID	Use
AXI read requests	b000	Single reads and device or strongly ordered burst transactions.
	b001	Non-cacheable pagewalks from the TLB.
	b010	Linefills and non-cacheable bursts from linefill buffer 0.
	b011	Linefills and non-cacheable bursts from linefill buffer 1.
	b100	Any instruction fetch.

Table 8-2 AXI ID signal encodings (continued)

Request type	ID	Use
AXI write requests	b00	Writes to normal non-cacheable memory, all SWPs, and all store-exclusive transactions.
	b01	Writes and evictions to non-shareable cacheable memory.
	b10	Writes to device and strongly ordered memory.

8.1.5 AXI user bits

Table 8-3 shows the bit encodings for **ARUSER[4:0]**

Table 8-3 ARUSER[4:0] encodings

Bits	Name	Description

[4:0]	Inner attributes	b00001 = Strongly ordered
		b00010 = Device, non-shareable
		b00011 = Device, shareable
		b00110 = Non-cacheable, non-shareable
		b00111 = Non-cacheable, shareable
		b11110 = Writeback cacheable, read and write allocate, non-shareable
		b11111 = Writeback cacheable, read and write allocate, shareable

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

Table 8-4 AWUSER[6:0] encodings

Bits	Name	Description

[6:5]	Exclusive mode	b00 = Not an eviction
		b01 = An eviction with dirty data
		b10 = Not used
		b11 = An eviction but the data is clean

[4:0]	Inner attributes	b00001 = Strongly ordered
		b00010 = Device, non-shareable
		b00011 = Device, shareable
		b00110 = Non-cacheable, non-shareable
		b00111 = Non-cacheable, shareable
		b11110 = Writeback cacheable, read and write allocate, non-shareable
		b11111 = Writeback cacheable, read and write allocate, shareable

Note

Table 8-3 and Table 8-4 show the attributes after they have been modified by the TLB remapping, and therefore represent the attributes used by the L1 cache, not necessarily the attributes that were stored in the page tables. If the L1 data cache is disabled, all cacheable memory is remapped to non-cacheable.

8.1.6 Write response

The AXI master requires that the slave does not return a write response until it has received the write address.

8.1.7 Exclusive L2 cache

The Cortex-A5 processor can be connected to an L2 cache that supports an exclusive cache mode. This mode must be activated both in the Cortex-A5 processor and in the L2 cache controller. See *Auxiliary Control Register* on page 4-41.

In this mode, the data cache of the 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 processor. When exclusive cache configuration is selected:

- Data cache line replacement policy is modified so that the victim line is always evicted to L2 memory, even if it is clean.
- If a line is dirty in the L2 cache controller, a read request to this address from the processor causes writeback to external memory and a linefill to the processor.

8.2 AXI privilege information

AXI provides information about the privilege level of an access on the **ARPROT** and **AWPROT** signals. However, when accesses might be cached or merged together, the resulting transaction can have both privileged and user data combined. If this happens, the Cortex-A5 processor marks the transaction as privileged, even it if was initiated by a user process.

Table 8-5 shows Cortex-A5 processor modes and corresponding **ARPROT[0]** and **AWPROT[0]** values.

Table 8-5 Cortex-A5 mode and APROT values

Processor mode	Type of access	Value of APROT
-	Cacheable read access	Privileged
User	Device, strongly ordered, or normal non-cacheable read access	User
Privileged		Privileged
-	Cacheable write access	Always marked as Privileged
User	Device or strongly ordered write	User
Privileged		Privileged
User	Normal non-cacheable write	Privileged, except for `SWP`, `SWPB`, `STREX`, `STREXB`, `STREXH`, and `STREXD`
Privileged	Normal non-cacheable write	Privileged
-	TLB pagewalk	Privileged

Chapter 9
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:

9.1 About debug

The Cortex-A5 processor implements the ARMv7 debug architecture, including support for the TrustZone security extensions and CoreSight. The processor forms one component of a debug system. Figure 9-1 shows a typical system.

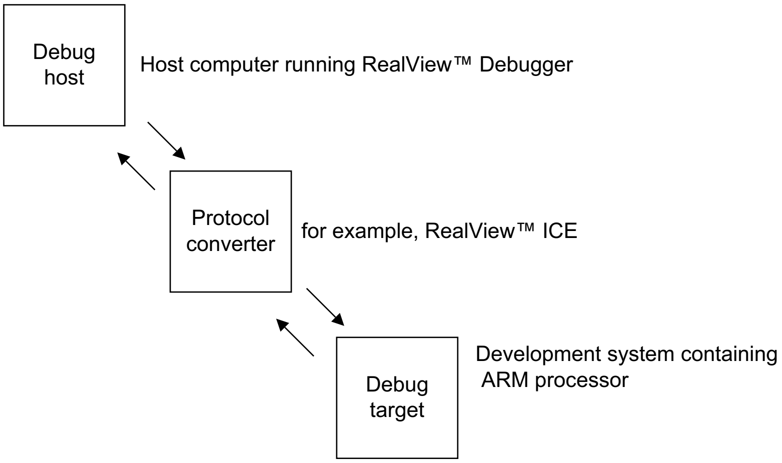

Debug host
Host computer running RealView™ Debugger
Protocol converter
for example, RealView™ ICE
Debug target
Development system containing ARM processor

Figure 9-1 Typical debug system

This typical system has a:

- debug host
- protocol converter
- debug target.

9.1.1 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.

9.1.2 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.

9.1.3 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-A5 Debug Unit using the APB slave port.

9.1.4 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 using breakpoints and a watchpoint
- examine and alter processor and coprocessor state
- examine and alter memory and input/output peripheral state
- restart the processor core and resume execution of the application.

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

- Halting debug-mode debugging
- Monitor debug-mode debugging
- trace debugging, see *ETM interface* on page 2-7.

9.2 Debugging modes

The Cortex-A5 processor implements the following 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.

DBGEN and **SPIDEN** control invasive debug permissions.

Noninvasive debug

Noninvasive debug is defined as a debug process where you can observe the processor but not control it. The *Embedded Trace Macrocell* (ETM) interface and the performance monitor registers are features of noninvasive debug.

See *ETM interface* on page 2-7 for information on the ETM interface.

See *System performance monitor registers* on page 4-7 for information on performance monitor registers.

NIDEN and **SPNIDEN** control non-invasive debug permissions. Non-invasive debug is always permitted when invasive debug is permitted.

The following sections describe:

- *Halting debug-mode debugging*
- *Monitor debug-mode debugging*
- *Performance monitor and events* on page 9-5
- *Security extensions and debugging* on page 9-5.

9.2.1 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.

9.2.2 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.

When execution of a monitor target starts, the state of the processor is preserved in the same way as all ARM exceptions. The monitor target then communicates with the debugger to access processor and coprocessor state, and to access memory contents and input/output peripherals. Monitor debug-mode requires a debug monitor program to interface between the debug hardware and the software debugger.

Debug can also be carried out using a trace-based approach with output from the external ETM interface, described in *ETM interface* on page 2-7.

The Cortex-A5 processor enables access to the internal debug functionality and registers as follows:

- through a memory-mapped area on the external AMBA APBv3 slave port

- by using CP14 system coprocessor operations from software running on the processor.

9.2.3 Performance monitor and events

The Cortex-A5 processor includes logic to detect various events that can occur during runtime. These events provide useful information about the behavior of the processor that you can use when debugging or profiling code.

The events are made visible on an output bus, **EVNTBUS**, which forms part of the ETM interface, and can also be counted using registers in the *Performance Monitoring Unit* (PMU). The Cortex-A5 processor enables access to the PMU registers as follows:

- through a memory-mapped area on the external AMBA APBv3 slave port
- by using CP15 system coprocessor operations from software running on the processor.

The Cortex-A5 processor supports two general purpose counters which can be tied to any of the events, and a separate dedicated cycle counter.

See *Event Type Select Register* on page 10-11 for more information on performance events.

9.2.4 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. In this configuration, only Monitor debug-mode debugging is supported in 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 the CP15 system coprocessor control the secure debug permissions.

See *Secure Debug Enable Register* on page 4-47, *Authentication signals* on page 9-43, and *Changing the authentication signals* on page 9-43 for more information.

9.3 Debug interface

The Cortex-A5 processor implements the ARMv7 debug architecture and debug events as described in the *ARM Architecture Reference Manual*.

9.3.1 Breakpoints and watchpoints

The Cortex-A5 processor supports three hardware breakpoints and two watchpoints. Two of the breakpoints match only to virtual address, the third matches against either virtual address or context ID. Similarly, either of the first two breakpoints can be linked to the third breakpoint to enable an instruction to be trapped in a given process context. The following apply to watchpoints:

- A watchpoint event is always synchronous. It has the same behavior as a synchronous data abort.
- If a translation or access permissions fault 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.

9.3.2 Asynchronous aborts

The Cortex-A5 processor ensures that all possible outstanding asynchronous data aborts have been recognized prior to entry to debug state.

9.3.3 Processor interfaces

The Cortex-A5 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 debug register map using the APB slave port. See *External debug interface* on page A-9.

Performance monitor

This interface is CP15 based and memory-mapped. You can access the performance monitor memory map using the APB slave port. See Chapter 10 *Performance Monitoring Unit* for information on performance monitor registers.

Trace registers

This interface is memory-mapped. The Cortex-A5 processor implements the ETM architecture. See *ETM interface* on page 2-7.

9.3.4 Effects of resets on debug registers

nCPURESET

The **nCPURESET** signal is the main processor reset that initializes the Cortex-A5 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**.

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.

9.4 Debug register summary

You can access the debug registers:

- through the CP14 interface. The debug registers are mapped to coprocessor instructions.
- through a 4KB memory-mapped region on the APB interface when **PADDRDBG[12]** is b0 and using the relevant offset, with the following exceptions:
  - — DBGRAR
  - — DBGSAR
  - — DBGDSCRint
  - — DBGDTRTXint
  - — DBGDTRRXint.

External views of DBGDSCR , DBGDTRRX, and DBGDTRTX are accessible through memory-mapped APB access.

Table 9-1 shows the debug interface registers, including the CP14 instruction encoding and the address offset in the 4KB memory mapped region on the APB interface.

Table 9-1 Debug interface registers

Register number	Offset	CP14 instruction	Access	Name	Description
0	`0x000`	`0 c0 c0 0`	RO	DBGDIDR[a]	*Debug Identification Register* on page 9-11 [b]
128	-	`0 c1 c0 0`	RO	DBGDRAR[a]	-
256	-	`0 c2 c0 0`	RO	DBGDSAR[a]	-
1	-	`0 c0 c1 0`	RO	DBGDSCR-int[ab]	*Debug Status and Control Register* on page 9-12 [a]
2-4	-	-	-	Reserved	-
5	-	`0 c0 c5 0`	RW	DBGDTRTXint (writes) and DBGDTRRXint (reads)[a]	-
6	`0x018`	`0 c0 c6 0`	RW	DBGWFAR	Use of DBGWFAR is deprecated in the ARMv7 architecture, because watchpoints are synchronous
7	`0x01C`	`0 c0 c7 0`	RW	DBGVCR	-
8	-	-	-	Reserved	-
9	`0x024`	`0 c0 c9 0`	RAZ/WI	DBGECR	*Event Catch Register* on page 9-19
10	`0x028`	`0 c0 c10 0`	RAZ/WI	DBGDSCCR	*Debug State Cache Control Register* on page 9-19
11	`0x02C`	`0 c0 c11 0`	RAZ/WI	DBGDSMCR	*Debug State MMU Control Register* on page 9-19
12-31	-	-	-	Reserved	-
32	`0x080`	`0 c0 c0 2`	RW	DBGDTRRXext	-
33	`0x084`	-	WO	DBGITR	-

Table 9-1 Debug interface registers (continued)

Register number	Offset	CP14 instruction	Access	Name	Description
33	`0x084`	-	RO	DBGPCSR	*Program Counter Sampling Register* on page 9-18
34	`0x088`	`0 c0 c2 2`	RW	DBGDSCRext	*Debug Status and Control Register* on page 9-12
35	`0x08C`	`0 c0 c3 2`	RW	DBGDTRTXext	-
36	`0x090`	`0 c0 c4 2`	WO	DBGDRCR	*Debug Run Control Register* on page 9-19
37-39	-	-	-	Reserved	-
40	`0x0A0`	-	RO	DBGPCSR	*Program Counter Sampling Register* on page 9-18
41	`0x0A4`	-	RO	DBGCIDSR	-
42-63	-	-	-	Reserved	-
64-79	`0x100-0x13C`	`0 c0 c0-2 4`	RW	DBGBVRn	*Breakpoint Value Registers* on page 9-20
80-95	`0x140-0x17C`	`0 c0 c0-2 5`	RW	DBGBCRn	*Breakpoint Control Registers* on page 9-21
96-111	`0x180-0x1BC`	`0 c0 c0-1 6`	RW	DBGWVRn	*Watchpoint Value Register* on page 9-24
129-255, 257-191	`0x1C0-0x1FC`	`0 c0 c0-1 7`	RW	DBGWCRn	*Watchpoint Control Register* on page 9-25
128-191	-	-	-	Reserved	-
192	`0x300`	`0 c1 c0 4`	WO/WI	DBGOSLAR	*Operating System Lock and Save/Restore Registers* on page 9-19
193	`0x304`	`0 c1 c1 4`	RO/RAZ	DBGOSLSR	
194	`0x308`	`0 c1 c2 4`	RAZ/WI	DBGOSSRR	
195	-	-	-	Reserved	-
196	`0x310`	`0 c1 c4 4`	RW	DBGPRCR	*Device Power-down and Reset Control Register* on page 9-26
197	`0x314`	`0 c1 c5 4`	RO	DBGPRSR	*Device Power-down and Reset Status Register* on page 9-28
198-831	-	-	-	Reserved	-
832-895	`0xD00-0xDFC`	-[c]	RO	-	*Processor ID Registers* on page 9-29
896-957	-	-	-	Reserved	-
958	`0xEF8`	-	WO	DBGITMISCOUT	*DBGITMISCOUT Register (Miscellaneous Outputs)* on page 9-38
959	`0xEFC`	-	RO	DBGITMISCIN	*DBGITMISCIN Register (Miscellaneous Inputs)* on page 9-39
960	`0xF00`	-	RAZ/WI	DBGITCTRL	*Integration Mode Control Register* on page 9-40
961-999	`0xF04-0xF9C`	-	-	Reserved	-
1000	`0xFA0`	`0 c7 c8 6`	RW	DBGCLAIMSET	*Claim Tag Set Register* on page 9-30

Table 9-1 Debug interface registers (continued)

Register number	Offset	CP14 instruction	Access	Name	Description
1001	`0xFA4`	`0 c7 c9 6`	RW	DBGCLAIMCLR	*Claim Tag Clear Register* on page 9-31
1002-1003	-	-	-	Reserved	-
1004	`0xFB0`	-	WO	DBGLAR	*Lock Access Register* on page 9-31
1005	`0xFB4`	-	RO	DBGLSR	*Lock Status Register* on page 9-32
1006	`0xFB8`	`0 c7 c14 6`	RO	DBGAUTHSTATUS	*Authentication Status Register* on page 9-33
1007-1009	-	-	-	Reserved	-
1010	`0xFC8`	`0 c7 c2 7`	RO	DBGDEVID	-
1011	`0xFCC`	-	RO	DBGDEVTYPE	*Device Type Register* on page 9-34
1012-1016	`0xFD0-0xFEC`	-	RO	PERIPHERALID	*Identification Registers* on page 9-34
1017-1019	-	-	-	Reserved	-
1020-1023	`0xFF0-0xFFC`	-	RO	COMPONENTID	*Identification Registers* on page 9-34

[a] Baseline CP14 interface. This register also has an external view through the memory-mapped interface and the CP14 interface. See *Debug Status and Control Register* on page 9-12.

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

[c] Accessed through CP15 interface.

9.5 Debug register descriptions

This section describes the debug registers.

9.5.1 Debug Identification Register

The DBGDIDR characteristics are:

Purpose Identifies the debug architecture version and specifies the number of debug resources that the Cortex-A5 processor implements.

Usage constraints There are no usage constraints.

Configurations Available in all configurations.

Attributes See the register summary in Table 9-1 on page 9-8.

Note

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

Figure 9-2 shows the DBGDIDR bit assignments.

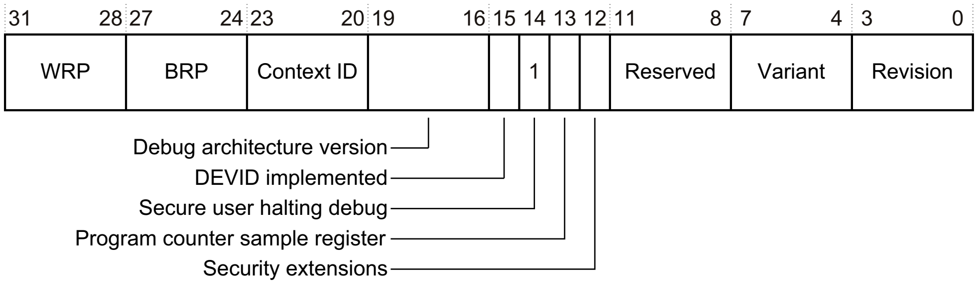


Figure 9-2 DBGDIDR bit assignments

Table 9-2 shows the DBGDIDR bit assignments.

Table 9-2 DBGDIDR bit assignments

Bits	Name	Description
[31:28]	WRP	Number of Watchpoint Register Pairs: For the Cortex-A5 processor, this field reads as b0001 to indicate two WRPs are implemented.
[27:24]	BRP	Number of Breakpoint Register Pairs: For the Cortex-A5 processor, this field reads as b0010 to indicate three BRPs are implemented.
[23:20]	Context	Number of Breakpoint Register Pairs with context ID comparison capability: For the Cortex-A5 processor, this field reads as b0000 to indicate one BRP has context ID capability.
[19:16]	Debug architecture version	Debug architecture version: b0011 = ARMv7 Debug with Extended CP14 interface implemented.
[15]	DEVID implemented	For the Cortex-A5 processor, this field reads as b1 to indicate that the Debug Device ID Register, DBGDEVID is implemented.

Table 9-2 DBGDIDR bit assignments (continued)

Bits	Name	Description
[14]	Secure User halting debug not supported	For the Cortex-A5 processor, this field reads as b1 to indicate that Secure User halting debug is not supported.
[13]	Program Counter Sampling Register	Program Counter Sample Register, DBGPCSR. For the Cortex-A5 processor, this field reads as b1 to indicate that DBGPCSR is implemented as debug register 33.
[12]	Security extensions	Security extensions bit: For the Cortex-A5 processor, this field reads as b1 to indicate that the debug security extensions are implemented.
[11:8]	Reserved	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 [3:0] of the ID Code Register in CP15 c0.

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

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

See *Main ID Register* on page 4-21 for a description of CP15 c0, Main ID Register.

To use the DBGDIDR, 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
```

9.5.2 Debug Status and Control Register

The DBGDSCR characteristics are:

Purpose Contains status and control information about the debug unit.

Usage constraints There are no usage constraints.

Configurations Available in all configurations.

Attributes See the register summary in Table 9-1 on page 9-8.

Figure 9-3 on page 9-13 shows the DBGDSCR bit assignments.

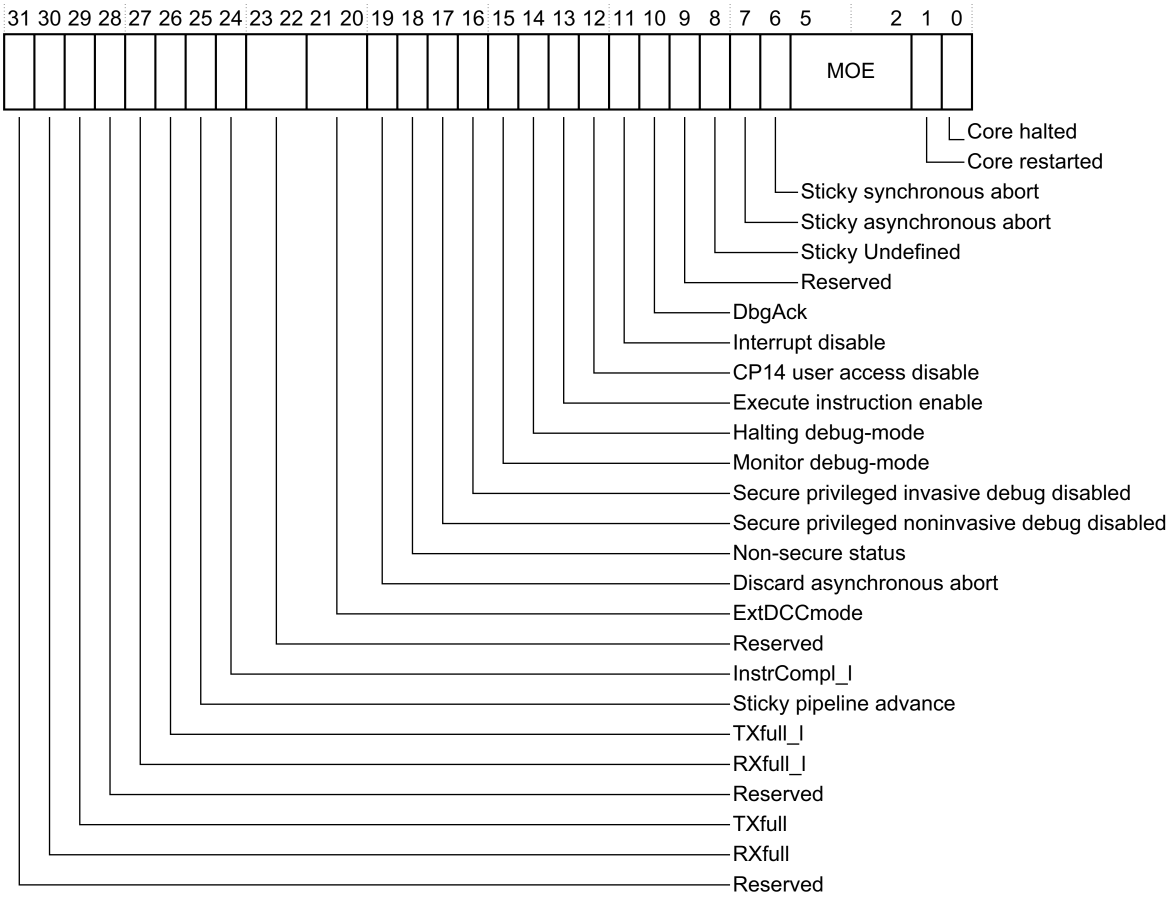


Figure 9-3 DBGDSCR bit assignments

Table 9-3 shows DBGDSCR bit assignments.

Table 9-3 DBGDSCR bit assignments

Bits	Name	Description
[31]	Reserved	RAZ on reads, SBZP on writes.
[30]	RXfull	The DBGDTRRX Register full flag: 0 = DBGDTRRX empty, reset value 1 = DBGDTRRX full. When set, this flag indicates that there is data available in the Receive Data Transfer Register, DBGDTRRX. It is automatically set on writes to the DBGDTRRXext by the debugger, and is cleared when the processor reads the CP14 DBGDTRRXint. If the flag is not set, reads of the DBGDTRRX return an Unpredictable value.
[29]	TXfull	The DBGDTRTX Register full flag: 0 = DBGDTRTX empty, reset value 1 = DBGDTRTX full. When clear, this flag indicates that the Transmit Data Transfer Register, DBGDTRTX is ready for data write. It is automatically cleared on reads of the DBGDTRTXext by the debugger, and is set when the processor writes to the CP14 DBGDTRTXint. If this bit is set and the processor attempts to write to the DBGDTRTXint, results are Unpredictable.
[28]	Reserved	RAZ on reads, SBZP on writes.

Table 9-3 DBGDSCR bit assignments (continued)

Bits	Name	Description
[27]	RXfull_l	The latched DBGDTRRX Register full flag. This flag is read in one of the following ways: • in DBGDSCRint using a CP14 instruction • in DBGDSCRext using the APB interface or CP14 instruction. Reads of DBGDSCRint return an Unpredictable value for this bit. Reads of DBGDSCRext return the same value as RXfull. If a write to the DBGDTRRXext address succeeds, RXfull_l is set to 1.
[26]	TXfull_l	The latched DBGDTRTX Register full flag. This flag is read in one of the following ways: • in DBGDSCRint using a CP14 instruction • in DBGDSCRext using the APB interface or CP14 instruction. Reads of DBGDSCRint return an Unpredictable value for this bit. Reads of DBGDSCRext return the same value as TXfull. If a read to the DBGDTRTXext address succeeds, TXfull_l is cleared.
[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 DBGDRCR[3] clears this bit. See *Debug Run Control Register* on page 9-19. 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: • in DBGDSCRint using CP14 instructions • in DBGDSCRext using the APB interface. When in Non-debug state, all reads of DBGDSCR return an Unpredictable value for this bit. Otherwise, reads through the CP14 interface return an Unpredictable value for this bit. Reads of the DBGDSCRext APB address return the same value as InstrCompl. If a write to the DBGITR APB address succeeds while in Stall or Nonblocking mode, InstrCompl_l and InstrCompl are cleared. If a write to the DBGDTRRXext APB address or a read to the DBGDTRTXext 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 interface. 0 = the processor is currently executing an instruction fetched from the DBGITR Register, reset value 1 = the processor is not currently executing an instruction fetched from the DBGITR Register.
[23:22]	Reserved	RAZ on reads, SBZP on writes.
[21:20]	ExtDCCmode	External DCC access mode. This is a read and write field. You can use this field to optimize DTR and DBGITR 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 DBGDSCR, DTR, and DBGITR 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 *External DCC and DBGITR access mode* on page 9-17 for more information.

Table 9-3 DBGDSCR 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* on page 9-33.
[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* on page 9-33.
[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, DBGDSCR[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, DBGDSCR[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 a DBGITR write succeeds, the processor fetches an instruction from the DBGITR for execution. If this bit is set to 1 when the processor is not in debug state, the behavior of the processor is Unpredictable.
[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.

Table 9-3 DBGDSCR bit assignments (continued)

Bits	Name	Description
[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. This bit is ignored when either: • DBGDSCR[15:14] == 0b00 • **DBGEN** is LOW.
[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]	Reserved	RAZ on reads, SBZP on writes.
[8]	Sticky Undefined	Sticky Undefined bit: 0 = No Undefined instruction exception occurred in debug state since the last time this bit was cleared. This is the reset value. 1 = An Undefined instruction exception has occurred while in debug state since the last time this bit was cleared. This flag detects Undefined instruction exceptions generated by instructions issued to the processor through the DBGITR. This bit is set to 1 when an Undefined instruction exception occurs while the processor is in debug state. Writing a 1 to DBGDRCR[2] clears this bit. See *Debug Run Control Register* on page 9-19.
[7]	Sticky asynchronous abort	Sticky asynchronous Data Abort bit: 0 = no asynchronous Aborts occurred since the last time this bit was cleared, reset value 1 = an asynchronous Abort occurred since the last time this bit was cleared. This flag detects asynchronous Aborts triggered by instructions issued to the processor through the DBGITR. This bit is set to 1 when an asynchronous Abort occurs while the processor is in debug state. Writing a 1 to DBGDRCR[2] clears this bit. See *Debug Run Control Register* on page 9-19.
[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 DBGITR. This bit is set to 1 when a synchronous Data Abort occurs while the processor is in debug state. Writing a 1 to DBGDRCR[2] clears this bit. See *Debug Run Control Register* on page 9-19. When this is set, no instructions are issued through the DBGITR. Writes to DBGITR are ignored and, if ExtDCCmode is configured for Fast mode, reads of DBGDTRTXext and writes of DBGDTRRXext are ignored.

Table 9-3 DBGDSCR bit assignments (continued)

Bits	Name	Description
[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 = a vector catch debug event occurred b1010 = a synchronous watchpoint debug event occurred 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 DBGDSCR[18] to a 0 and DBGDSCR[1:0] to b10.

Internal view

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

To access the DBGDSCRint, 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 DBGDSCRext through the Extended CP14 interface, read or write CP14 c2 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
```

External DCC and DBGITR access mode

You can use the DBGDSCR.ExtDCCmode field to optimize data transfer between a debugger and the processor.

The DBGDSCR.ExtDCCmode can be one of the following:

- Nonblocking. This is the default mode

- Stall
- Fast.

In Nonblocking mode, the APB reads from the DBGDTRTXext and writes to the DBGDTRRXext and DBGITR are ignored if the appropriate latched READY flag is not in the ready state. In particular:

- writes to DBGDTRRXext are ignored if RXfull_l is set
- writes to DBGITR are ignored if InstrCompl_l is not set
- reads from DBGDTRTXext are ignored and return an Unpredictable value if TXfull_l is not set.

Debuggers accessing these registers must first read DBGDSCRext. This has the side-effect of copying RXfull and TXfull to RXfull_l and TXfull_l, and setting InstrCompl_l. The debugger can then use the returned value to determine whether a subsequent access to these registers will be ignored.

In Stall mode, the APB accesses to DBGDTRRXext, DBGDTRTXext, and DBGITR stall under the following conditions:

- writes to DBGDTRRXext are stalled until RXfull is cleared
- writes to DBGITR are stalled until InstrCompl is set
- reads from DBGDTRTXext are stalled until TXfull is set.

In Fast mode, the processor fetches an instruction from the DBGITR when a DBGDTRRXext write or DBGDTRTXext read succeeds. In Stall mode and Nonblocking mode, the processor fetches an instruction from the DBGITR when an DBGITR write succeeds.

9.5.3 Program Counter Sampling Register

The DBGPCSR characteristics are:

Purpose 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.

Usage constraints There are no usage constraints.

Configurations Available in all configurations.

Attributes See the register summary in Table 9-1 on page 9-8.

Figure 9-4 shows the DBGPCSR bit assignments.

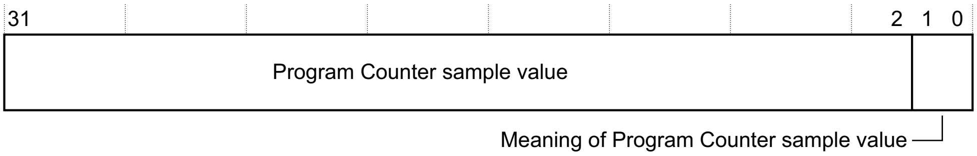

31
2
1
0
Program Counter sample value
Meaning of Program Counter sample value

Figure 9-4 DBGPCSR bit assignments

Table 9-4 shows the DBGPCSR bit assignments.

Table 9-4 DBGPCSR bit assignments

Bits	Name	Description
[31:2]	Program Counter Sample value	The sampled value of bits [31:2] of the Program Counter.

[1:0]	Meaning of Program Counter Sample value	b00 = References an ARM state instruction.
		bx1 = References a Thumb or ThumbEE state instruction.
		b10 = Jazelle-DBX.

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

9.5.4 Debug State Cache Control Register

The DBGDSCCR controls cache behavior while the processor is in debug state. The Cortex-A5 processor does not implement any of the features of the DBGDSCCR. The DBGDSCCR is read-as-zero.

9.5.5 Event Catch Register

The DBGECR configures the debug logic to generate a debug event in certain circumstances. The Cortex-A5 processor does not implement any of the features of the DBGECR. The DBGECR is read-as-zero.

9.5.6 Debug State MMU Control Register

The DBGDSMCR controls MMU behavior while the processor is in debug state. The Cortex-A5 processor does not implement any of the features of the DBGDSMCR. The DBGDSMCR is read-as-zero.

9.5.7 Operating System Lock and Save/Restore Registers

The DBGOSLAR, DBGOSLSR, and DBGOSSRR registers are defined in the ARMv7 Debug architecture to allow access control and save/restore of operating system registers during debug. The Cortex-A5 processor does not implement any of the features of these registers. The DBGOSSRR is read-as-zero and write-ignore, DBGOSLSR is read-only and read-as-zero, and DBGOSLAR is write-only and write ignore.

9.5.8 Debug Run Control Register

The DBGDRCR characteristics are:

Purpose Requests the processor to enter or leave debug state. It also clears the sticky exception bits present in the DBGDSCR.

Usage constraints There are no usage constraints.

Configurations Available in all configurations.

Attributes See the register summary in Table 9-1 on page 9-8.

Figure 9-5 on page 9-20 shows the DBGDRCR bit assignments.

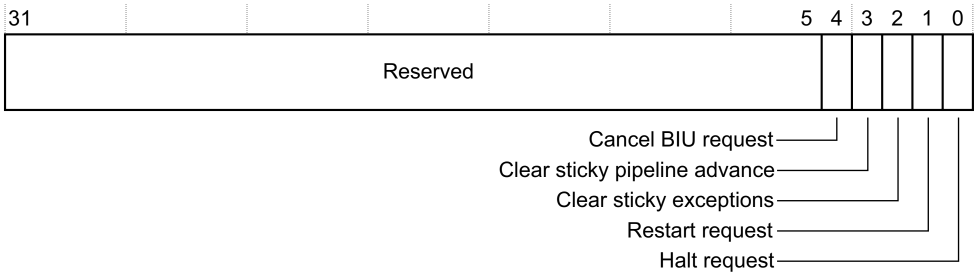


Figure 9-5 DBGDRCR bit assignments

Table 9-5 shows the DBGDRCR bit assignments.

Table 9-5 DBGDRCR bit assignments

Bits	Name	Description
[31:5]	Reserved	RAZ/SBZP.
[4]	Cancel BIU request	Implemented.
[3]	Clear sticky pipeline advance	Clear sticky pipeline advance. Writing a 1 to this bit clears DBGDSCR[25].
[2]	Clear sticky exceptions	Clear sticky exceptions. Writing a 1 to this bit clears DBGDSCR[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 DBGDSCR[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 DBGDSCR[0] until it reads 1. This bit always reads as zero. Writes are ignored when the processor is already in debug state.

9.5.9 Breakpoint Value Registers

The DBGBVR characteristics are:

Purpose Contains the breakpoint value that corresponds to either an instruction address or a context ID. Breakpoints can be set on:

- an instruction address
- a context ID value
- an instruction address and context ID pair.

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

Usage constraints There are no usage constraints.

Configurations Available in all configurations.

Attributes See the register summary in Table 9-1 on page 9-8.

Each DBGBVR is associated with a *Breakpoint Control Register* (DBGBCR), as follows:

- DBGBVR0 with DBGBCR0
- DBGBVR1 with DBGBCR1

- DBGBVR2 with DBGBCR2.

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

Table 9-6 shows the DBGBVRs and corresponding DBGBCRs.

Table 9-6 DBGBVRs and corresponding DBGBCRs

Breakpoint Value Registers			Breakpoint Control Registers		
Register	Register number	Offset	Register	Register number	Offset
DBGBVR0	64	`0x100`	DBGBCR0	80	`0x140`
DBGBVR1	65	`0x104`	DBGBCR1	81	`0x144`
DBGBVR2	66	`0x108`	DBGBCR2	82	`0x148`

Table 9-7 shows the DBGBVR bit assignments.

Table 9-7 DBGBVR bit assignments

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

Note

- Only BRP2 supports context ID comparison.
- DBGBVR0[1:0] and DBGBVR1[1:0] are SBZP on writes and RAZ on reads because these registers do not support context ID comparisons.
- The context ID value for DBGBVR2 to match with is given by the contents of the CP15 Context ID Register. See Chapter 4 *System Control* for information on the Context ID Register.

9.5.10 Breakpoint Control Registers

The DBGBCR characteristics are:

Purpose Contains the necessary control bits for setting breakpoints and linked breakpoints.

Usage constraints There are no usage constraints.

Configurations Available in all configurations.

Attributes See the register summary in Table 9-1 on page 9-8.

Figure 9-6 on page 9-22 shows the DBGBCR bit assignments.

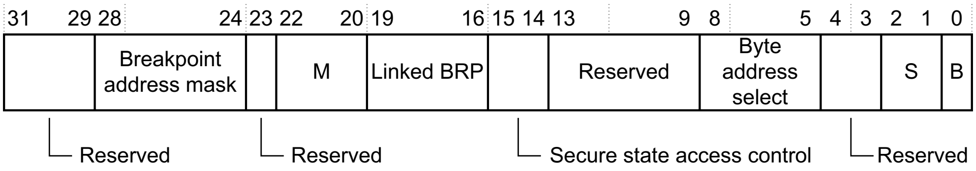


Figure 9-6 DBGBCR bit assignments

Table 9-8 shows the DBGBCR bit assignments.

Table 9-8 DBGBCR bit assignments

Bits	Name	Description
[31:29]	Reserved	RAZ on reads, SBZP on writes.
[28:24]	Breakpoint address mask	Breakpoint address mask. RAZ/WI b00000 = no mask
[23]	Reserved	RAZ on reads, SBZP on writes.
[22:20]	M	Meaning of DBGBVR: 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** DBGBCR0[21] and DBGBCR1[21] are RAZ on reads because these registers do not have context ID comparison capability.
[19:16]	Linked BRP	Linked BRP number. The binary number encoded here indicates another BRP to link this one with. **Note** • If a BRP is linked with itself, it is Unpredictable whether a breakpoint debug event is generated • If this BRP is linked to another BRP that is not configured for linked context ID matching, it is Unpredictable whether a breakpoint debug event is generated.
[15:14]	Secure state access control	Secure state access control. This field enables the breakpoint to be conditional on the security state of the processor. b00 = breakpoint matches in both Secure and Non-secure state b01 = breakpoint only matches in Non-secure state b10 = breakpoint only matches in Secure state b11 = reserved.
[13:9]	Reserved	RAZ on reads, SBZP on writes.

Table 9-8 DBGBCR bit assignments (continued)

Bits	Name	Description
[8:5]	Byte address select	Byte address select. For breakpoints programmed to match an instruction address, you must write a word-aligned address to the DBGBVR. 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 instruction address match: b0000 = the breakpoint never hits b0011 = the breakpoint hits if any of the two bytes starting at address DBGBVR & `0xFFFFFFFC` +0 is accessed b1100 = the breakpoint hits if any of the two bytes starting at address DBGBVR & `0xFFFFFFFC` +2 is accessed b1111 = the breakpoint hits if any of the four bytes starting at address DBGBVR & `0xFFFFFFFC` +0 is accessed. If you program the BRP for instruction address mismatch, the breakpoint hits where the corresponding instruction address breakpoint does not hit, that is, the range of addresses covered by an instruction address mismatch breakpoint is the negative image of the corresponding instruction address 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.
[4:3]	Reserved	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 9-9 Meaning of DBGBVR bits [22:20]

BVR[22:20]	Meaning
b000	The corresponding DBGBVR[31:2] is compared against the instruction address bus and the state of the processor against this DBGBCR. It generates a breakpoint debug event on a joint instruction address and state match.
b001	The corresponding DBGBVR[31:2] is compared against the instruction address bus and the state of the processor against this DBGBCR. This BRP is linked with the one indicated by DBGBCR[19:16] linked BRP field. They generate a breakpoint debug event on a joint instruction address, context ID, and state match.
b010	The corresponding DBGBVR[31:0] is compared against CP15 Context ID Register, c13 and the state of the processor against this DBGBCR. 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, DBGBCR[8:5] must be set to b1111. Otherwise, it is Unpredictable whether a breakpoint debug event is generated.
b011	The corresponding DBGBVR[31:0] is compared against CP15 Context ID Register, c13. This BRP links another BRP (of the DBGBCR[21:20]=b01 type), or WRP (with DBGWCR[20]=b1). They generate a breakpoint or watchpoint debug event on a joint instruction address or data address and context ID match. For this BRP, DBGBCR[8:5] must be set to b1111, DBGBCR[15:14] must be set to b00, and DBGBCR[2:1] must be set to b11. Otherwise, it is Unpredictable whether a breakpoint debug event is generated.

Table 9-9 Meaning of DBGBVR bits [22:20] (continued)

BVR[22:20]	Meaning
b100	The corresponding DBGBVR[31:2] and DBGBCR[8:5] are compared against the instruction address bus and the state of the processor against this DBGBCR. It generates a breakpoint debug event on a joint instruction address mismatch and state match.
b101	The corresponding DBGBVR[31:2] and DBGBCR[8:5] are compared against the instruction address bus and the state of the processor against this DBGBCR. This BRP is linked with the one indicated by DBGBCR[19:16] linked BRP field. It generates a breakpoint debug event on a joint instruction address mismatch, state and context ID match.
b11x	Reserved. The behavior is Unpredictable.

9.5.11 Watchpoint Value Register

The DBGWVR characteristics are:

Purpose Contains the watchpoint value that corresponds to a data address and can be set either on:

- a data address
- a data address and context ID pair.

For a data address 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 data address and the context ID pair match simultaneously.

Usage constraints There are no usage constraints.

Configurations Available in all configurations.

Attributes See the register summary in Table 9-1 on page 9-8.

The DBGWVR is associated with a *Watchpoint Control Register* (DBGWCR).

Table 9-10 shows the DBGWVR and DBGWCR relationship.

Table 9-10 WVR and corresponding WCR

Watchpoint Value Registers			Watchpoint Control Registers		
Register	Register number	Offset	Register	Register number	Offset
DBGWVR0	96	`0x180`	DBGWCR0	112	`0x1C0`
DBGWVR1	97	`0x184`	DBGWCR1	113	`0x1C4`

A pair of watchpoint registers, DBGWVR and DBGWCR, is called a *Watchpoint Register Pair* (WRP).

Table 9-11 shows the DBGWVR bit assignments.

Table 9-11 DBGWVR bit assignments

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

9.5.12 Watchpoint Control Register

The DBGWCR characteristics are:

Purpose Contains the necessary control bits for setting watchpoints and linked watchpoints.

Usage constraints There are no usage constraints.

Configurations Available in all configurations.

Attributes See the register summary in Table 9-1 on page 9-8.

Figure 9-7 shows the DBGWCR bit assignments.

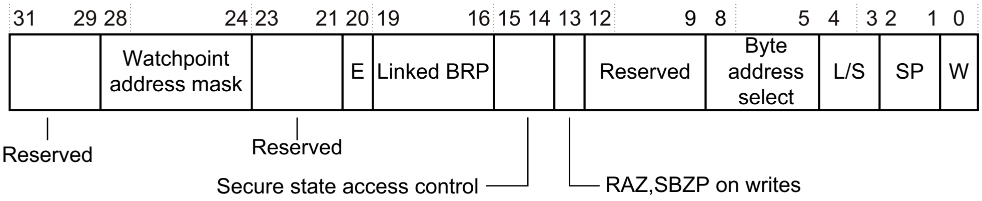


Figure 9-7 DBGWCR bit assignments

Table 9-12 shows the DBGWCR bit assignments.

Table 9-12 DBGWCR bit assignments

Bits	Name	Description
[31:29]	Reserved	RAZ on reads, SBZP on writes.
[28:24]	Watchpoint address mask	This field watches a range of addresses by masking lower order address bits out of the watchpoint comparison: b00000 = no mask b00001 = reserved b00010 = reserved b00011 = `0x00000007` mask for data address b00100 = `0x0000000F` mask for data address b00101 = `0x0000001F` mask for data address . . . b11111 = `0x7FFFFFFF` mask for data address. **Note** • If bits [28:24] are not set to b00000, bits [12:5] must be set to b11111111. Otherwise the behavior is Unpredictable. • If [28:24] are not set to b00000, the corresponding DBGWVR bits that are not being included in the comparison SBZ. Otherwise the behavior is Unpredictable. To watch for a write to any byte in an 8-byte aligned object of size 8 bytes, ARM recommends that a debugger sets bits [28:24] to b00111, and bits [12:5] to b11111111. This is compatible with both ARMv7 debug compliant implementations that have an 8-bit byte address select field (bits [12:5]) and with those that have a 4-bit byte address select field (bits [8:5]).
[23:21]	Reserved	RAZ on reads, SBZP on writes.

Table 9-12 DBGWCR bit assignments (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]	Reserved	RAZ on reads, SBZP on writes.
[12:9]	Reserved	RAZ/WI
[8:5]	Byte address select	Byte address select. The DBGWVR is programmed with word-aligned address. You can use this field to program the watchpoint so it only hits if certain byte addresses are accessed.
[4:3]	L/S	Load/store access. The watchpoint can be conditioned to the type of access being done. b00 = reserved b01 = load, load exclusive, or swap b10 = store, store exclusive or swap b11 = either. SWP and SWPB trigger a watchpoint on b01, b10, or b11. A load exclusive instruction triggers a watchpoint on b01 or b11. A store exclusive instruction triggers a watchpoint on b10 or b11 only if it passes the local monitor within the processor.[a]
[2:1]	S	Privileged access control[b]. 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.
[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.

[b] For all cases, the match refers to the privilege of the access, not the mode of the processor.

9.5.13 Device Power-down and Reset Control Register

The DBGPRCR characteristics are:

Purpose Controls power-down related functionality.

Usage constraints There are no usage constraints.

Configurations Available in all configurations.

Attributes See the register summary in Table 9-1 on page 9-8.

Figure 9-8 shows the DBGPRCR bit assignments.

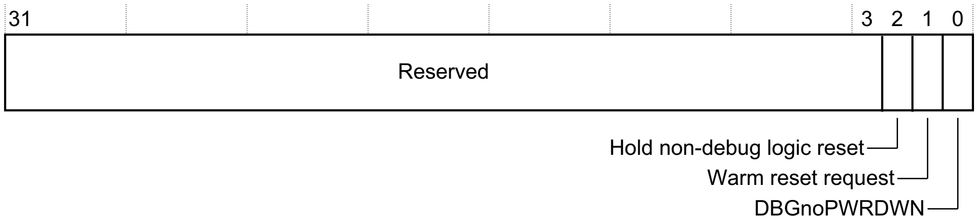


Figure 9-8 DBGPRCR bit assignments

Table 9-13 shows the DBGPRCR bit assignments.

Table 9-13 DBGPRCR bit assignments

Bits	Name	Description
[31:3]	Reserved	RAZ on reads, SBZP on writes
[2]	Hold non-debug logic reset	Hold non-debug logic reset: 0 = Do not hold the non-debug logic reset on power-up or warm reset. 1 = Hold the non-debug logic of the processor in reset on power-up or warm reset. The processor is held in this state until this flag is cleared to 0.
[1]	Warm reset request	Reset request bit. Writing 1 to this bit generates a warm reset: 0 = No action. 1 = Request internal reset. This bit generates an internal warm reset of the non-debug logic. ——— **Note** ——— • This bit is always RAZ. Software must read the Sticky Reset status bit in the DBGPRSR to determine the current reset status of the processor. • Warm reset request does not request the reset of any registers that are only reset on a debug logic reset. The external debugger can use this bit to force the processor into reset if it does not have access to the **nCPURESET** input. The reset behavior is the same as warm reset driven by the **nCPURESET** signal. A warm reset does not cause power-down. The effect of this bit depends on the state of the external debug interface signals. — A write to this bit is ignored unless both the external debug interface signals **DBGEN** and **SPIDEN** are HIGH, meaning that invasive debug is permitted in all processor states and modes. — Both the Warm reset request and Hold non-debug logic reset bits can be set to 1 in a single write to the DBGPRCR. In this case the processor enters reset and is held there. See the *ARM Architecture Reference Manual* for more information.
[0]	DBGnoPWRDWN	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-A5 processor and ETM 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.

9.5.14 Device Power-down and Reset Status Register

The DBGPRSR characteristics are:

Purpose Provides information about the reset and power-down state of the processor.

Usage constraints There are no usage constraints.

Configurations Available in all configurations.

Attributes See the register summary in Table 9-1 on page 9-8.

——— **Note** ———

The Cortex-A5 processor does not support debug of powered down processors.

————————

Figure 9-9 shows the DBGPRSR bit assignments.

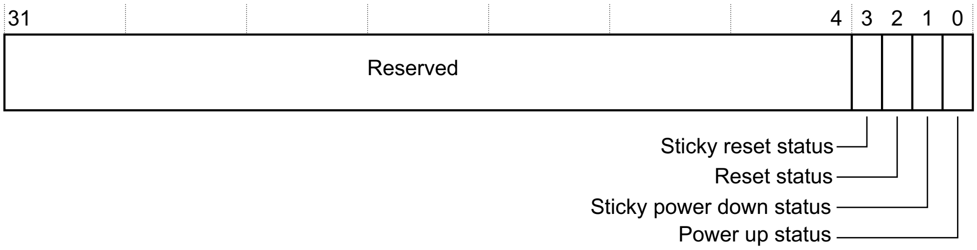


Figure 9-9 DBGPRSR bit assignments

Table 9-14 shows the DBGPRSR bit assignments.

Table 9-14 DBGPRSR bit assignments

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

9.6 Management registers

The management registers define the standardized set of registers implemented by all CoreSight components. These registers are described in this section.

Table 9-15 shows the contents of the management registers for the Cortex-A5 processor debug unit.

Table 9-15 Management registers

Offset	Register number	Access	Mnemonic	Description
`0xD00-0xDFC`	832-895	RO	-	*Processor ID Registers*
`0xE00-0xEF0`	854-957	-	-	RAZ
`0xF00`	960	RW	DBGITCTRL	*Integration Mode Control Register* on page 9-40.
`0xF04-0xF9C`	961-999	RAZ	-	Reserved for Management Register expansion
`0xFA0`	1000	RW	DBGCLAIMSET	*Claim Tag Set Register* on page 9-30
`0xFA4`	1001	RW	DBGCLAIMCLR	*Claim Tag Clear Register* on page 9-31
`0xFA8-0xFBC`	1002-1003	-	-	RAZ
`0xFB0`	1004	WO	DBGLAR	*Lock Access Register* on page 9-31
`0xFB4`	1005	RO	DBGLSR	*Lock Status Register* on page 9-32
`0xFB8`	1006	RO	DBGAUTHSTATUS	*Authentication Status Register* on page 9-33
`0xFBC-0xFC4`	1007-1009	-	-	RAZ
`0xFC8`	1010	RO	DBGDEVID	Device Identifier.
`0xFCC`	1011	RO	DBGDEVTYPE	*Device Type Register* on page 9-34
`0xFD0-0xFFC`	1012-1023	RO	-	*Identification Registers* on page 9-34

9.6.1 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 9-16 shows the offset value, register number, mnemonic, and description that are associated with each Processor ID Register.

Table 9-16 Processor Identifier Registers

Offset (hex)	Register number	Access	Mnemonic	Reset value	Description
`0xD00`	832	RO	MIDR	`0x410FC051`	*Main ID Register* on page 4-21
`0xD04`	833	RO	CTR	`0x83338003`	*Cache Type Register* on page 4-21
`0xD08`	834	RAZ	-	-	-
`0xD0C`	835	RO	TLBTR	`0x00000000`	*TLB Type Register* on page 4-23
`0xD10`	836	-	-	-	Reserved

Table 9-16 Processor Identifier Registers (continued)

Offset (hex)	Register number	Access	Mnemonic	Reset value	Description
`0XD14`	837	RO	MPIDR	-	*Multiprocessor Affinity Register* on page 4-23
`0xD18-0xD1C`	838-839	RO	-	`0x410FC051`	Alias of MIDR
`0xD20`	840	RO	ID_PFR0	`0x00001231`	Processor Feature Register 0
`0xD24`	841	RO	ID_PFR1	`0x00000011`	Processor Feature Register 1
`0xD28`	842	RO	ID_DFR0	`0x02010444`	Debug Feature Register 0
`0xD2C`	843	RAZ	ID_AFR0	-	Auxiliary Feature Register 0
`0xD30`	844	RO	ID_MMFR0	`0x00100103`	Memory Model Feature Register 0
`0xD34`	845	RO	ID_MMFR1	`0x40000000`	Memory Model Feature Register 1
`0xD38`	846	RO	ID_MMFR2	`0x01230000`	Memory Model Feature Register 2
`0xD3C`	847	RO	ID_MMFR3	`0x00102211`	Memory Model Feature Register 3
`0xD40`	848	RO	ID_ISAR0	`0x00101111`	Instruction Set Attribute Register 0
`0xD44`	849	RO	ID_ISAR1	`0x13112111`	Instruction Set Attribute Register 1
`0xD48`	850	RO	ID_ISAR2	`0x21232041`	Instruction Set Attribute Register 2
`0xD4C`	851	RO	ID_ISAR3	`0x11112131`	Instruction Set Attribute Register 3
`0xD50`	852	RO	ID_ISAR4	`0x00011142`	Instruction Set Attribute Register 4
`0xD54`	853	RAZ	ID_ISAR5	-	Instruction Set Attribute Register 5

9.6.2 Claim Tag Set Register

The DBGCLAIMSET Register characteristics are:

Purpose 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.

Usage constraints There are no usage constraints.

Configurations Available in all configurations.

Attributes See the register summary in Table 9-15 on page 9-29.

Figure 9-10 shows the DBGCLAIMSET Register bit assignments.

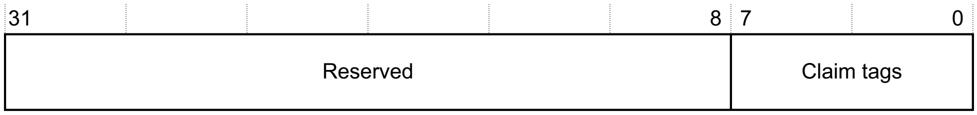


Figure 9-10 DBGCLAIMSET Register bit assignments

Table 9-17 shows the DBGCLAIMSET Register bit assignments.

Table 9-17 DBGCLAIMSET Register bit assignments

Bits	Name	Description
[31:8]	Reserved	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.

9.6.3 Claim Tag Clear Register

The DBGCLAIMCLR characteristics are:

Purpose Read the claim status on debug resources.

Usage constraints There are no usage constraints.

Configurations Available in all configurations.

Attributes See the register summary in Table 9-15 on page 9-29.

Figure 9-11 shows the DBGCLAIMCLR bit assignments.

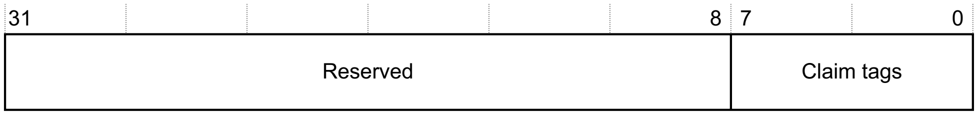


Figure 9-11 DBGCLAIMCLR bit assignments

Table 9-18 shows the DBGCLAIMCLR bit assignments.

Table 9-18 DBGCLAIMCLR bit assignments

Bits	Name	Description
[31:8]	Reserved	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.

9.6.4 Lock Access Register

The DBGLAR characteristics are:

Purpose Controls writes to the debug registers. This reduces 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 register is in the debug power domain, it is not lost when the core powers down.

Usage constraints There are no usage constraints.

Configurations Available in all configurations.

Attributes See the register summary in Table 9-15 on page 9-29.

Figure 9-12 shows the DBGLAR bit assignments.

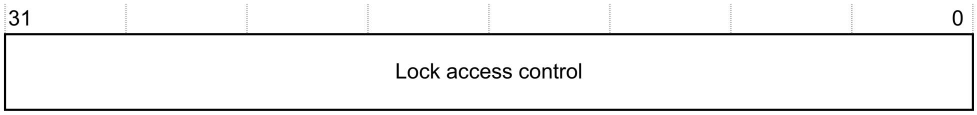


Figure 9-12 DBGLAR bit assignments

Table 9-19 shows the DBGLAR bit assignments.

Table 9-19 DBGLAR 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 write with **PADDRDBG31**=1, the results are Unpredictable.

9.6.5 Lock Status Register

The DBGLSR characteristics are:

Purpose Returns the current lock status of the debug registers.

Usage constraints There are no usage constraints.

Configurations Available in all configurations.

Attributes See the register summary in Table 9-15 on page 9-29.

Figure 9-13 shows the DBGLSR bit assignments.

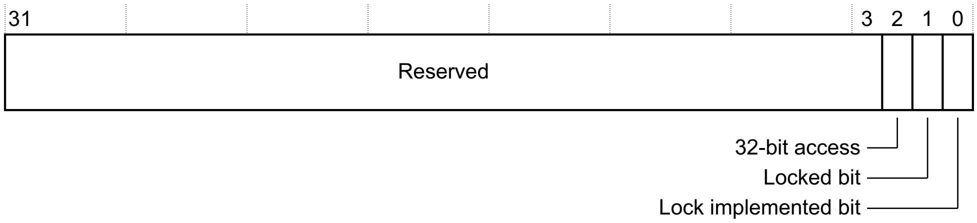


Figure 9-13 DBSLSR bit assignments

Table 9-20 shows the DBGLSR bit assignments.

Table 9-20 DBGLSR bit assignments

Bits	Name	Description
[31:3]	Reserved	Reserved

Table 9-20 DBGLSR bit assignments (continued)

Bits	Name	Description
[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`.

9.6.6 Authentication Status Register

The DBGAUTHSTATUS Register characteristics are:

Purpose Reads the current values of the configuration inputs that determine the debug permission level.

Usage constraints There are no usage constraints.

Configurations Available in all configurations.

Attributes See the register summary in Table 9-15 on page 9-29.

Figure 9-14 shows the DBGAUTHSTATUS Register bit assignments.

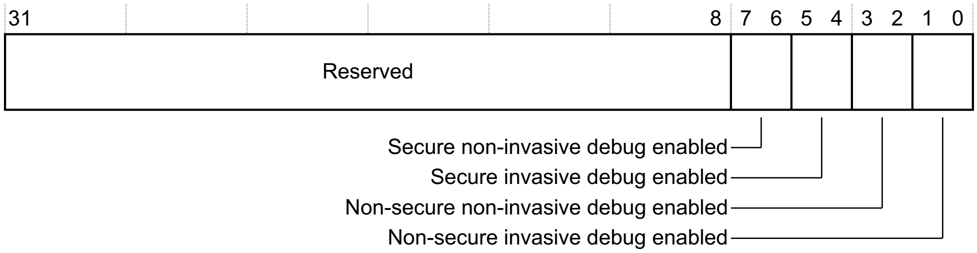

31
8
7
6
5
4
3
2
1
0
Reserved
Secure non-invasive debug enabled
Secure invasive debug enabled
Non-secure non-invasive debug enabled
Non-secure invasive debug enabled

Figure 9-14 DBGAUTHSTATUS Register bit assignments

Table 9-21 shows the DBGAUTHSTATUS Register bit assignments.

Table 9-21 DBGAUTHSTATUS Register bit assignments

Bits	Name	Value	Description
[31:8]	Reserved	-	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**	

Table 9-21 DBGAUTHSTATUS Register bit assignments (continued)

Bits	Name	Value	Description
[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**	

9.6.7 Device Type Register

The DBGDEVTYPE Register characteristics are:

Purpose Indicates the type of debug component.

Usage constraints There are no usage constraints.

Configurations Available in all configurations.

Attributes See the register summary in Table 9-15 on page 9-29.

Figure 9-15 shows the DBGDEVTYPE Register bit assignments.

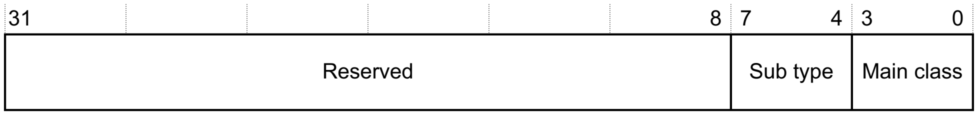


Figure 9-15 DBGDEVTYPE Register bit assignments

Table 9-22 shows the DBGDEVTYPE Register bit assignments.

Table 9-22 DBGDEVTYPE Register bit assignments

Bits	Name	Description
[31:8]	Reserved	RAZ.
[7:4]	Sub type	Indicates that the sub-type of the Cortex-A5 processor is *core*. This value is `0x1`.
[3:0]	Main class	Indicates that the main class of the Cortex-A5 processor is *debug logic*. This value is `0x5`.

9.6.8 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 9-23 shows the offset value, register number, and description that are associated with each Peripheral Identification Register.

Table 9-23 Peripheral Identification Registers

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

Table 9-24 shows the Peripheral ID Register 0 bit assignments.

Table 9-24 Peripheral ID Register 0 bit assignments

Bits	Description
[31:8]	RAZ.
[7:0]	Indicates bits [7:0] of the part number for the Cortex-A5 processor. This value is `0x05`.

Table 9-25 shows the Peripheral ID Register 1 bit assignments.

Table 9-25 Peripheral ID Register 1 bit assignments

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-A5 processor. This value is `0xC`.

Table 9-26 shows the Peripheral ID Register 2 bit assignments.

Table 9-26 Peripheral ID Register 2 bit assignments

Bits	Description
[31:8]	RAZ.
[7:4]	Indicates the revision number for the Cortex-A5 processor. This value changes based on the product major and minor revision. This value is set to 1 indicating revision r0p1.
[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 9-27 shows the Peripheral ID Register 3 bit assignments.

Table 9-27 Peripheral ID Register 3 bit assignments

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-A5 processor, this value is set to 0.

Table 9-28 shows the Peripheral ID Register 4 bit assignments.

Table 9-28 Peripheral ID Register 4 bit assignments

Bits	Description
[31:8]	RAZ.
[7:4]	Indicates the number of blocks occupied by the Cortex-A5 processor. This field is always set to 0.
[3:0]	Indicates the JEDEC JEP106 Continuation Code. For the Cortex-A5 processor, this value is `0x4`.

Table 9-29 shows the offset value, register number, and value that are associated with each Component Identification Register.

Table 9-29 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

9.7 Integration test registers

The Cortex-A5 processor contains Integration Test Registers that enable you to verify integration of the design and enable topology detection of the design using debug tools. The *Integration Mode Control Register* on page 9-40 controls the use of the Integration Test Registers.

When programming the Integration Test Registers you must enable all the changes at the same time.

For more information about the Integration Test Registers and the Integration Mode Control Register see the *ARM Architecture Reference Manual*.

9.7.1 Processor integration testing

This section describes the behavior and use of the Integration Test Registers that are in the processor. It also describes the Integration Mode Control Register that controls the use of the Integration Test Registers. For more information about the DBGITCTRL see the *ARM Architecture Reference Manual*.

If you want to access these registers you must first set bit [0] of the Integration Mode Control Register to 1.

- You can use the write-only Integration Test Registers to set the outputs of some of the processor signals. Table 9-30 shows the signals that you can write in this way.
- You can use the read-only Integration Test Registers to read the state of some of the processor inputs. Table 9-31 shows the signals that you can read in this way.

Table 9-30 Output signals that can be controlled by the Integration Test Registers

Signal	Register	Bit	Register description
DBGRESTARTED	DBGITMISCOUT	[9]	See *DBGITMISCOUT Register (Miscellaneous Outputs)* on page 9-38
PMUIRQ	DBGITMISCOUT	[4]	
DBGACK	DBGITMISCOUT	[0]	

Table 9-31 Input signals that can be read by the Integration Test Registers

Signal	Register	Bit	Register description
DBGRESTART	DBGITMISCIN	[11]	See *DBGITMISCIN Register (Miscellaneous Inputs)* on page 9-39
nIRQ	DBGITMISCIN	[2]	
nFIQ	DBGITMISCIN	[1]	
EDBGRQ	DBGITMISCIN	[0]	

This section describes:

- *Using the Integration Test Registers* on page 9-38
- *Performing integration testing* on page 9-38
- *DBGITMISCOUT Register (Miscellaneous Outputs)* on page 9-38
- *DBGITMISCIN Register (Miscellaneous Inputs)* on page 9-39
- *Integration Mode Control Register* on page 9-40.

Using the Integration Test Registers

When bit [0] of the Integration Mode Control Register (DBGITCTRL) is set to b1:

- Values written to the write-only Integration Test Registers map onto the specified outputs of the processor. For example, writing b1 to DBGITMISCOUT[0] causes **DBGACK** to be asserted HIGH.
- Values read from the read-only Integration Test Registers correspond to the values of the specified inputs of the processor. For example, if you read DBGITMISCIN[11] you obtain the value of **DBGRESTART**.

Performing integration testing

When you perform integration testing or topology detection, ARM strongly recommends that the processor is halted while in debug state, because toggling input and output pins might have an unwanted effect on the operation of the processor. You must not set the DBGITCTRL Register until the processor has halted.

After you perform integration testing or topology detection, that is, the Integration Mode Control Register has been set, the system must be reset. This is because the signals that are toggled can have an unwanted effect on connected devices.

DBGITMISCOUT Register (Miscellaneous Outputs)

The DBGITMISCOUT Register characteristics are:

Purpose Sets the state of the output pins shown in Table 9-30 on page 9-37.

Usage constraints
- Available when bit [0] of DBGITCTRL is set to 1
- The value of the register sets the signals on the output pins when the register is written.

Configurations Available in all configurations.

Attributes See the register summary in Table 9-1 on page 9-8.

Figure 9-16 shows the DBGITMISCOUT Register bit assignments.

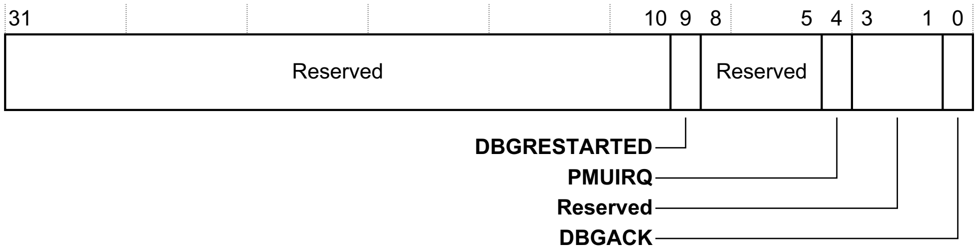


Figure 9-16 DBGITMISCOUT Register bit assignments

Table 9-32 shows the DBGITMISCOUT Register bit assignments. When this register is written the appropriate output pins take the value written.

Table 9-32 DBGITMISCOUT Register bit assignments

Bits	Name	Function
[31:10]	-	Reserved. Write as zero.
[9]	DBGRESTARTED	Set value of the **DBGRESTARTED** output pin.
[8:5]	-	Reserved. Write as zero.
[4]	PMUIRQ	Set value of **PMUIRQ** output pin.
[3:1]	-	Reserved. Write as zero.
[0]	DBGACK	Set value of the **DBGACK** output pin.

DBGITMISCIN Register (Miscellaneous Inputs)

The DBGITMISCIN Register characteristics are:

Purpose Reads the state of the input pins shown in Table 9-31 on page 9-37.

Usage constraints
- Available when bit [0] of DBGITCTRL is set to 1
- The values of the register bits depend on the signals on the input pins when the register is read.

Configurations Available in all configurations.

Attributes See the register summary in Table 9-1 on page 9-8.

Figure 9-17 shows the DBGITMISCIN Register bit assignments.

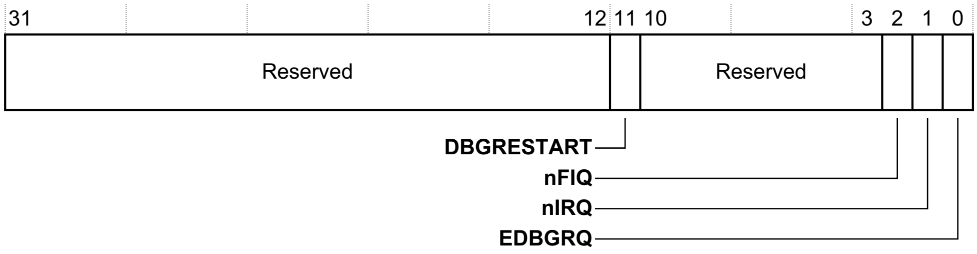


Figure 9-17 DBGITMISCIN Register bit assignments

Table 9-33 shows the DBGITMISCIN Register bit assignments.

Table 9-33 DBGITMISCIN Register bit assignments

Bits	Name	Function
[31:12]	-	Reserved. Read Undefined.
[11]	DBGRESTART	Read value of the **DBGRESTART** input pin.
[10:3]	-	Reserved. Read Undefined.

Table 9-33 DBGITMISCIN Register bit assignments (continued)

Bits	Name	Function
[2]	nFIQ	Read value of **nFIQ** input pin.
[1]	nIRQ	Read value of **nIRQ** input pin.
[0]	EDBGRQ	Read value of **EDBGRQ** input pin.

Integration Mode Control Register

The DBGITCTRL Register characteristics are:

Purpose 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.

Usage constraints There are no usage constraints.

Configurations Available in all configurations.

Attributes See the register summary in Table 9-1 on page 9-8.

Figure 9-18 shows the DBGITCTRL Register bit assignments.

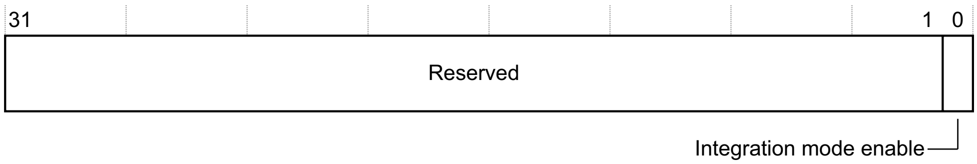


Figure 9-18 DBGITCTRL Register bit assignments

Table 9-34 shows the DBGITCTRL Register bit assignments.

Table 9-34 DBGITCTRL Register bit assignments

Bits	Name	Function
[31:1]	-	Reserved.

[0]	INTMODE	Controls whether the processor is in normal operating mode or integration mode:
		b0 = normal operation
		b1 = integration mode enabled.

Writing to the DBGITCTRL register controls whether the processor is in its default functional mode, or in integration mode, where the inputs and outputs of the device can be directly controlled for the purpose of integration testing or topology detection. For more information see the *ARM Architecture Reference Manual*.

9.8 External debug interface

The system can access memory-mapped debug registers through the Cortex-A5 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 lower 4KB region is used to access the debug registers. The upper 4KB is used to access the Performance Monitor Registers, see Chapter 10 *Performance Monitoring Unit*. The **PADDRDBG31** signal indicates to the processor the source of access. See *External debug interface* on page A-9 for a complete list of the external debug signals.

Figure 9-19 shows the external debug interface signals.

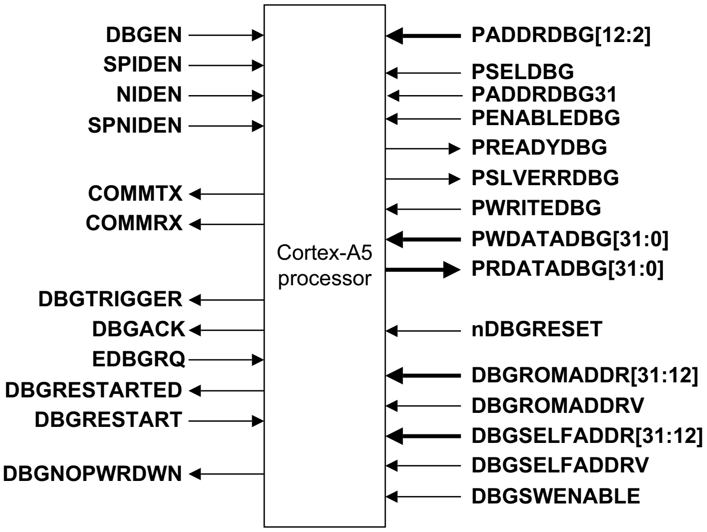


Figure 9-19 External debug interface signals

9.9 Miscellaneous debug signals

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

9.9.1 EDBGRQ

This signal generates a halting debug event, that is, it requests the processor to enter debug state. When this occurs, the DBGDSCR[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 causes Unpredictable behavior.

9.9.2 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 DBGDSCR[10] DbgAck bit to 1. See *Debug Status and Control Register* on page 9-12.

9.9.3 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 DBGDSCR[30] DTRRX full flag. See *Debug Status and Control Register* on page 9-12.

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 DBGDSCR[29] DTRTX full flag. See *Debug Status and Control Register* on page 9-12.

9.9.4 Memory mapped accesses, DBGROMADDR, and DBGSELFADDR

The Cortex-A5 processor has a memory-mapped debug interface. If the system includes a suitable route from the AXI master interface to the APB debug slave interface, the processor can access the debug and PMU registers by executing load and store instructions:

- **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.

9.9.5 Authentication signals

Table 9-35 shows a list of the valid combination of authentication signals along with their 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 9-35 Authentication signal restrictions

SPIDEN	DBGEN[a]	SPNIDEN	NIDEN	Secure[b] invasive debug permitted	Non-secure invasive debug permitted	Secure non-invasive debug permitted	Non-secure non-invasive debug permitted
0	0	0	0	No	No	No	No
0	0	0	1	No	No	No	Yes
0	0	1	0	No	No	No	No
0	0	1	1	No	No	Yes	Yes
0	1	0	0	No	Yes	No	Yes
0	1	0	1	No	Yes	No	Yes
0	1	1	0	No	Yes	Yes	Yes
0	1	1	1	No	Yes	Yes	Yes
1	0	0	0	No	No	No	No
1	0	0	1	No	No	Yes	Yes
1	0	1	0	No	No	No	No
1	0	1	1	No	No	Yes	Yes
1	1	0	0	Yes	Yes	Yes	Yes
1	1	0	1	Yes	Yes	Yes	Yes
1	1	1	0	Yes	Yes	Yes	Yes
1	1	1	1	Yes	Yes	Yes	Yes

[a] When **DBGEN** is LOW, the processor behaves as if DBGDSCR[15:14] equals b00 with the exception that halting debug events are ignored when this signal is LOW.

[b] Invasive debug is defined as those operations that affect the behavior of the core. For example, taking a breakpoint is defined as invasive debug but performance counters and trace are noninvasive.

9.9.6 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-A5 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 DBGDSCR or DBGAUTHSTATUS 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 an ISB.

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 DBGDSCR[17:16], DBGDSCR[15:14], or DBGAUTHSTATUS.

Chapter 10
Performance Monitoring Unit

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

- *About the Performance Monitoring Unit* on page 10-2
- *Performance monitoring register descriptions* on page 10-3.

10.1 About the Performance Monitoring Unit

The Cortex-A5 processor PMU provides two counters to gather statistics on the operation of the processor and memory system. Each counter can count any of the events available in the Cortex-A5 processor. It also provides a single 32-bit cycle counter with support for scaling and filtering on the processor mode and security state. See the *ARM Architecture Reference Manual, ARMv7-A and ARMv7-R edition* and *ARM Architecture Reference Manual Performance Monitors v2 Supplement* for more information about performance monitoring.

The PMU counters, and their associated control registers, are accessible from the internal CP15 interface as well as through a 4KB memory mapped region on the APB interface when **PADDRDBG[12]** is b1. Table 10-1 shows the mappings of the PMU registers.

Table 10-1 Performance monitoring instructions and APB mapping

APB interface mapping	CP15 instruction	Access	Reset	Name	Description
`0x1000`	`0 c9 c13 2`	RW	-	PMXEVCNTR0	PM0 Counter Register
`0x1004`	`0 c9 c13 2`	RW	-	PMXEVCNTR1	PM1 Counter Register
`0x107C`	`0 c9 c13 0`	RW	-	PMCCNTR	Cycle Count Register
`0x1400`	`0 c9 c13 1`	RW	-	PMXEVTYPER0	PM0 Event Type Register
`0x1404`	`0 c9 c13 1`	RW	-	PMXEVTYPER1	PM0 Event Type Register
`0x147C`	`0 c9 c13 1`	RW	-	PMCCFILTR	Cycle Count Filter Control Register
`0x1C00`	`0 c9 c12 1`	RW	`0x00000000`	PMCNTENSET	Count Enable Set Register
`0x1C20`	`0 c9 c12 2`	RW	`0x00000000`	PMCNTENCLR	Count Enable Clear Register
`0x1C40`	`0 c9 c14 1`	RW	`0x00000000`	PMINTENSET	Interrupt Enable Set Register
`0x1C60`	`0 c9 c14 2`	RW	`0x00000000`	PMINTENCLR	Interrupt Enable Clear Register
`0x1C80`	`0 c9 c12 3`	RW	-	PMOVSR	Overflow Flag Status Register
`0x1CA0`	`0 c9 c12 4`	WO	-	PMSWINC	Software Increment Register
`0x1E00`	-	RO	`0x0009DF02`	PMCFGR	Configuration Register
`0x1E04`	`0 c9 c12 0`	RW	`0x41052000`	PMCR	Control Register
`0x1E08`	`0 c9 c14 0`	RW[a]	`0x00000000`	PMUSERENR	User Enable Register
`0x1E20`	`0 c9 c12 6`	RO	`0x003FFFFF`	PMCEID0	Common Event Identification Register 0
`0x1E24`	`0 c9 c12 7`	RO	`0x00000000`	PMCEID1	Common Event Identification Register 1
`0x1FB0`	-	WO	-	PMLAR	Lock Access Register
`0x1FB4`	-	RO	-	PMLSR	Lock Status Register
`0x1FB8`	-	RO	-	PMAUTHSTATUS	Authentication Status Register
`0x1FCC`	-	RO	-	PMDEVTYPE	Device Type Register
`0x1FD0-0x1FEC`	-	RO	-	PMPERIPHERALID	Identification Registers
`0x1FF0-0x1FFC`	-	RO	-	PMCOMPONENTID	Identification Registers

[a] Read only in user mode.

10.2 Performance monitoring register descriptions

This section describes the performance monitoring registers:

- *Performance Monitor Control Register*
- *Count Enable Set Register* on page 10-4
- *Count Enable Clear Register* on page 10-5
- *Overflow Flag Status Register* on page 10-6
- *Software Increment Register* on page 10-7
- *Event Counter Selection Register* on page 10-8
- *Common Event Identification Registers* on page 10-9
- *Cycle Count Register* on page 10-10
- *Event Type Select Register* on page 10-11
- *Cycle Count Filter Control Register* on page 10-13
- *Event Count Registers* on page 10-14
- *User Enable Register* on page 10-15
- *Interrupt Enable Set Register* on page 10-16
- *Interrupt Enable Clear Register* on page 10-16
- *Configuration Register* on page 10-17
- *Lock Access Register* on page 10-18
- *Lock Status Register* on page 10-19
- *Authentication Status Register* on page 10-20
- *Device Type Register* on page 10-20
- *Identification Registers* on page 10-21.

10.2.1 Performance Monitor Control Register

The PMCR characteristics are:

Purpose Controls operation of the:
- Performance Monitor Count Registers
- Cycle Counter Register.

Usage constraints The PMCR is:
- accessible as determined by the *User Enable Register* on page 10-15
- common to Secure and Non-secure states.

Configurations Available in all configurations.

Attributes See the register summary in Table 4-17 on page 4-18.

Figure 10-1 shows the PMCR bit assignments.

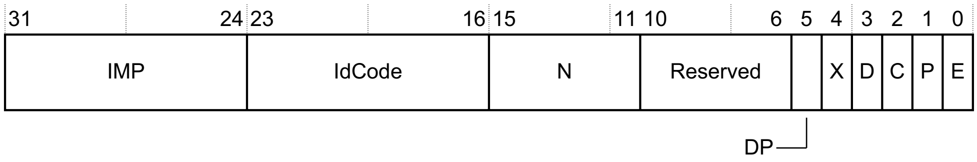


Figure 10-1 PMCR bit assignments

Table 10-2 shows the PMCR bit assignments.

Table 10-2 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: `0x5` This field is read-only and write ignored.
[15:11]	N	Specifies the number of counters implemented: `0x2` = two counters implemented. This field is read-only and write ignored.
[10:6]	Reserved	RAZ/SBZP
[5]	DP	Disables cycle counter, PMCCNTR, when prohibited: 0 = count is enabled in prohibited regions. This is the reset value. 1 = count is disabled in prohibited regions.
[4]	X	Enables export of the events from the event bus to an external monitoring block, such as an ETM: 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, PMCCNTR, to zero.
[1]	P	Performance counter reset, write only bit, RAZ: 0 = no action 1 = reset all performance counters to zero, not including PMCCNTR.
[0]	E	Enable bit: 0 = disable all counters, including PMCCNTR. This is the reset value. 1 = enable all counters including PMCCNTR.

The PMCR is always accessible in privileged modes.

To access the PMCR, use:

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

10.2.2 Count Enable Set Register

The PMCNTENSET characteristics are:

Purpose The PMCNTENSET Register:

- enables the PMCCNTR Register
- indicates which counters are enabled.

Usage constraints The PMCNTENSET Register is:

- accessible as determined by the *User Enable Register* on page 10-15
- common to Secure and Non-secure states.

Configurations Available in all configurations.

Attributes See the register summary in Table 4-17 on page 4-18.

Figure 10-2 shows the PMCNTENSET Register bit assignments.

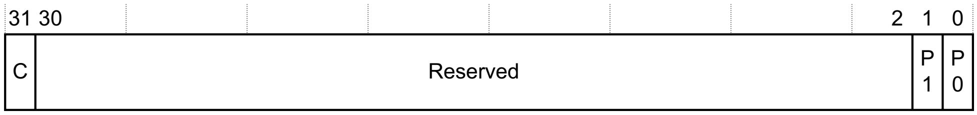


Figure 10-2 PMCNTENSET Register bit assignments

Table 10-3 shows the PMCNTENSET Register bit assignments.

Table 10-3 PMCNTENSET Register bit assignments

Bits	Name	Description
[31]	C	Cycle counter enable set: 0 = disable 1 = enable.
[30:2]	Reserved	RAZ/WI
[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 enables the counter.

10.2.3 Count Enable Clear Register

The PMCNTENCLR Register characteristics are:

Purpose Disables:

- the PMCCNTR Register.
- any implemented event counters.

Indicates counters that are enabled.

Usage constraints The PMCNTENCLR Register is:

- accessible as determined by the *User Enable Register* on page 10-15
- common to Secure and Non-secure states.

Configurations Available in all configurations.

Attributes See the register summary in Table 4-17 on page 4-18.

Figure 10-3 shows the PMCNTENCLR Register bit assignments.

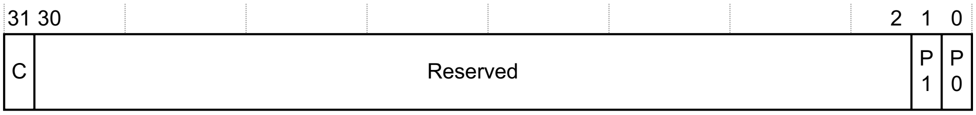


Figure 10-3 PMCNTENCLR Register bit assignments

Table 10-4 shows the PMCNTENCLR Register bit assignments.

Table 10-4 PMCNTENCLR Register bit assignments

Bits	Name	Description
[31]	C	Cycle counter enable clear: 0 = disable 1 = enable.
[30:2]	Reserved	RAZ/WI
[1]	P1	Counter 1 enable
[0]	P0	Counter 0 enable

To access the PMCNTENCLR Register, 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 disables the counter.

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 PMCR to disable all performance counters including PMCCNTR. The PMCNTENCLR Register retains its value when the enable bit of the PMCR is clear, even though its settings are ignored.

10.2.4 Overflow Flag Status Register

The PMSOVSR characteristics are:

Purpose Holds the state of the overflow flags for:
- the Cycle Count Register.
- each of the implemented event counters.

Usage constraints The PMSOVSR is:
- accessible as determined by the *User Enable Register* on page 10-15
- common to Secure and Non-secure states.

Configurations Available in all configurations.

Attributes See the register summary in Table 4-17 on page 4-18.

Figure 10-4 on page 10-7 shows the PMSOVSR bit assignments.

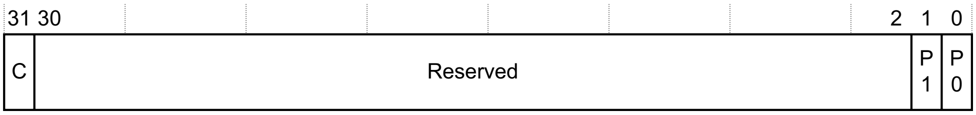


Figure 10-4 PMSOVSR bit assignments

Table 10-5 shows the PMSOVSR bit assignments.

Table 10-5 PMSOVSR Register bit assignments

Bits	Name	Description
[31]	C	Cycle counter overflow flag: 0 = disable 1 = enable.
[30:2]	Reserved	RAZ/WI
[1]	P1	Counter 1 overflow flag
[0]	P0	Counter 0 overflow flag

To access the PMSOVSR , use:

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

10.2.5 Software Increment Register

The PMSWINC Register characteristics are:

Purpose Increments the count of a performance monitor count register.

Usage constraints The PMSWINC Register is:

- accessible as determined by the *User Enable Register* on page 10-15
- common to Secure and Non-secure states.

Configurations Available in all configurations.

Attributes See the register summary in Table 4-17 on page 4-18.

Figure 10-5 shows the PMSWINC Register bit assignments.

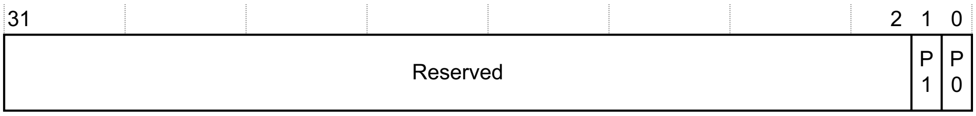


Figure 10-5 PMSWINC Register bit assignments

Table 10-6 shows the PMSWINC Register bit assignments.

Table 10-6 PMSWINC Register bit assignments

Bits	Name	Description
[31:2]	Reserved	SBZ
[1]	P1	Increment Counter 1
[0]	P0	Increment Counter 0

The PMSWINC Register 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 SWINC Register
```

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

10.2.6 Event Counter Selection Register

The PMSELR characteristics are:

Purpose Selects a Performance Monitor Count Register.

Usage constraints The PMSELR is:

- accessible as determined by the *User Enable Register* on page 10-15
- common to Secure and Non-secure states.

Configurations Available in all configurations.

Attributes See the register summary in Table 4-17 on page 4-18.

Figure 10-6 shows the PMSELR bit assignments.

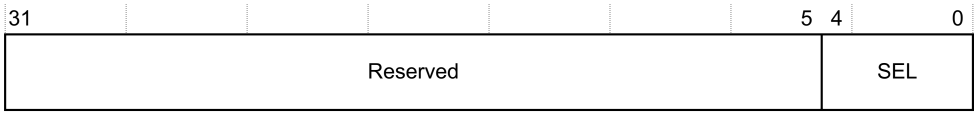


Figure 10-6 PMSELR bit assignments

Table 10-7 shows the PMSELR bit assignments.

Table 10-7 PMSELR bit assignments

Bits	Name	Description
[31:5]	Reserved	RAZ/SBZP

[4:0]	SEL	Counter select:
		b00000 = select counter 0
		b00001 = select counter 1
		b11111 = access the *Cycle Count Filter Control Register* on page 10-13.
		Note
		When these bits are set to b11111, reads and writes to the *Event Type Select Register* on page 10-11 are Unpredictable.

Values programmed in the PMSELR other than those specified in Table 10-7 on page 10-8 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
```

10.2.7 Common Event Identification Registers

The PMCEID0 and PMCEID1 Register characteristics are:

Purpose Define which common architectural and common micro-architectural feature events are implemented.

Usage constraints The PMCEID0 and PMCEID1 Registers are:

- accessible as determined by the *User Enable Register* on page 10-15
- common to Secure and Non-secure states.

Configurations Available in all configurations.

Attributes See the register summary in Table 4-17 on page 4-18.

Table 10-8 shows the PMCEID0 Register bit assignments and which features are implemented in the Cortex-A5 processor.

Table 10-8 PMCEID0 Register bit assignments

Bit	Number	Description	Implemented?
[31:30]	`0x1F-0x1E`	Reserved, UNK.	-
[29]	`0x1D`	Bus cycle	No
[28]	`0x1C`	Write to translation table base	No
[27]	`0x1B`	Instruction speculatively executed	No
[26]	`0x1A`	Local memory error	No
[25]	`0x19`	Bus access	No
[24]	`0x18`	Level 2 data cache write-back	No
[23]	`0x17`	Level 2 data cache refill	No
[22]	`0x16`	Level 2 data cache access	No
[21]	`0x15`	Level 1 data cache write-back	Yes
[20]	`0x14`	Level 1 instruction cache access	Yes
[19]	`0x13`	Data memory access	Yes
[18]	`0x12`	Predictable branch speculatively executed	Yes
[17]	`0x11`	Cycle	Yes
[16]	`0x10`	Mispredicted or not predicted branch speculatively executed	Yes
[15]	`0x0F`	Unaligned load or store	Yes
[14]	`0x0E`	Procedure return	Yes
[13]	`0x0D`	Immediate branch	Yes

Table 10-8 PMCEID0 Register bit assignments (continued)

Bit	Number	Description	Implemented?
[12]	`0x0C`	Software change of the PC	Yes
[11]	`0x0B`	Write to CONTEXTIDR	Yes
[10]	`0x0A`	Exception return	Yes
[9]	`0x09`	Exception taken	Yes
[8]	`0x08`	Instruction architecturally executed	Yes
[7]	`0x07`	Store	Yes
[6]	`0x06`	Load	Yes
[5]	`0x05`	Level 1 data TLB refill	Yes
[4]	`0x04`	Level 1 data cache access	Yes
[3]	`0x03`	Level 1 data cache refill	Yes
[2]	`0x02`	Level 1 instruction TLB refill	Yes
[1]	`0x01`	Level 1 instruction cache refill	Yes
[0]	`0x00`	Software increment	Yes

The PMCEID1 Register is reserved in the architecture, and is RAZ/WI.

To access the PMCEID0 Register, use:

```
MRC p15, 0, <Rd>, c9, c12, 6; Read PMCEID0
```

To access the PMCEID1 Register, use:

```
MRC p15, 0, <Rd>, c9, c12, 7; Read PMCEID1
```

10.2.8 Cycle Count Register

The PMCCNTR characteristics are:

Purpose Counts processor clock cycles.

Usage constraints The PMCCNTR is:

- accessible as determined by the *User Enable Register* on page 10-15
- common to Secure and Non-secure states.

Configurations Available in all configurations.

Attributes See the register summary in Table 4-17 on page 4-18.

To access the PMCCNTR, use:

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

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

10.2.9 Event Type Select Register

The PMXEVTYPER characteristics are:

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

Usage constraints The PMXEVTYPER is:

- accessible as determined by the *User Enable Register* on page 10-15
- common to Secure and Non-secure states.

Configurations Available in all configurations.

Attributes See the register summary in Table 4-17 on page 4-18.

Figure 10-7 shows the PMXEVTYPER bit assignments.

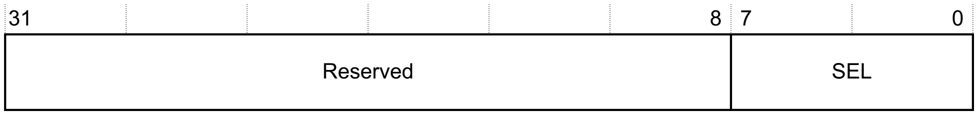


Figure 10-7 PMXEVTYPER bit assignments

Table 10-9 shows the PMXEVTYPER bit assignments.

Table 10-9 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*.

To access the PMXEVTYPER, use:

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

Table 10-10 shows the events that are generated, with the bit position of each event on the event bus, and the numbers that the PMU uses to refer the events. Event reference numbers that are not listed are reserved. The *ARM Architecture Reference Manual* shows the range of values for predefined events that you can monitor using the PMXEVTYPER.

Table 10-10 Performance monitor events

Event	EVNTBUS bit position	Description
`0x00`	-	Software increment. The register is incremented only on writes to the Software Increment Register. See *Software Increment Register* on page 10-7.
`0x01`	[0]	Instruction fetch that causes a refill at (at least) the lowest level of instruction or unified cache. Includes the speculative linefills in the count.
`0x02`	[1]	Instruction fetch that causes a TLB refill at (at least) the lowest level of TLB. Includes the speculative requests in the count.
`0x03`	[2]	Data read or write operation that causes a refill at (at least) the lowest level of data or unified cache. Counts the number of allocations performed in the Data Cache because of a read or a write.

Table 10-10 Performance monitor events (continued)

Event	EVNTBUS bit position	Description
`0x04`	[3]	Data read or write operation that causes a cache access at (at least) the lowest level of data or unified cache. This includes speculative reads.
`0x05`	[4]	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 because of PLD, PLI, CP15 Cache operation by MVA and CP15 VA to PA operations.
`0x06`	[5]	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`, and the reads because of the `SWP` instructions.
`0x07`	[6]	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`, and the writes because of the `SWP` instructions.
`0x08`	[7]	Instruction architecturally executed.
`0x09`	[8]	Exception taken. Counts the number of exceptions architecturally taken.
`0x0A`	[9]	Exception return architecturally executed. The following instructions are reported on this event: • `LDM {..., pc}^` • `RFE` • `DP S pc`
`0x0B`	[10]	Change to ContextID retired. Counts the number of instructions architecturally executed writing into the ContextID Register.
`0x0C`	[11]	Software change of PC.
`0x0D`	[12]	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`	[13]	Procedure return (other than exception returns) architecturally executed.
`0x0F`	[14]	Unaligned load-store.
`0x10`	[15]	Branch mispredicted/not predicted. Counts the number of mispredicted or not-predicted branches executed. This includes the branches which are flushed because of a previous load/store which aborts late.
`0x11`	-	Cycle counter.
`0x12`	[16]	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 because of a previous load/store which aborts late.
`0x13`	[17]	Data memory access.
`0x14`	[18]	Instruction Cache access.
`0x15`	[19]	Data cache eviction.
`0x86`	[20]	IRQ exception taken.
`0x87`	[21]	FIQ exception taken.
`0xC0`	[22]	External memory request.
`0xC1`	[23]	Non-cacheable external memory request.
`0xC2`	[24]	Linefill because of prefetch.

Table 10-10 Performance monitor events (continued)

Event	EVNTBUS bit position	Description
`0xC3`	[25]	Prefetch linefill dropped.
`0xC4`	[26]	Entering read allocate mode.
`0xC5`	[27]	Read allocate mode.
`0xC6`	[28]	Reserved.
`0xC7`	-	ETM Ext Out[0].
`0xC8`	-	ETM Ext Out[1].
`0xC9`	[29]	Data Write operation that stalls the pipeline because the store buffer is full.

If this unit generates an interrupt, the processor asserts the **PMUIRQ** signal. You can route this signal to an external interrupt controller for prioritization and masking.

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.

10.2.10 Cycle Count Filter Control Register

The PMCCFILTR characteristics are:

Purpose Configures which modes and states the Performance Monitor Count Register counts in.

Usage constraints The PMCCFILTR is:

- accessible as determined by the *Event Counter Selection Register* on page 10-8
- common to Secure and Non-secure states

Configurations Available in all configurations.

Attributes See the register summary in Table 4-17 on page 4-18.

Figure 10-8 shows the PMCCFILTR bit assignments.

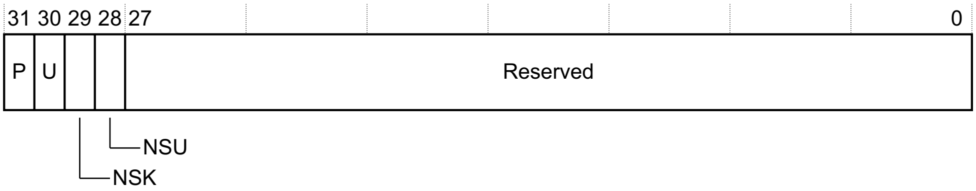


Figure 10-8 PMCCFILTR bit assignments

Table 10-11 shows the PMCCFILTR bit assignments.

Table 10-11 PMCCFILTR bit assignments

Bits	Name	Description
[31]	P	Configures the modes and states as described in the *ARM Architecture Reference Manual Performance Monitors v2 Supplement*
[30]	U	
[29]	NSK	
[28]	NSU	
[27:0]	-	UNK/SBZP

To access the PMCCFILTR, use:

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

10.2.11 Event Count Registers

The PMXEVCNTR characteristics are:

Purpose Count instances of an event selected by PMXEVTYPER. The bits of each PMXEVCNTR contain an event count.

Usage constraints The PMXEVCNTR are:

- accessible as determined by the *User Enable Register* on page 10-15
- common to Secure and Non-secure states.

Configurations There are two PMXEVCNTR.

Attributes See the register summary in Table 4-17 on page 4-18.

To access the PMXEVCNTR, use:

```
MRC p15, 0, <Rd>, c9, c13, 2 ; Read PMXEVCNTR0-PMXEVCNTR1 Registers
MCR p15, 0, <Rd>, c9, c13, 2 ; Write PMXEVCNTR0-PMXEVCNTR1 Registers
```

Table 10-12 shows what signal settings are required and the Secure or Non-secure state and mode that you can enable the counters.

Table 10-12 Signal settings for the PMXEVCNTR

DBGEN	SPIDEN	SDER.SUNIDEN	Secure state	User mode	PMCR[5]	Performance counters enabled	PMCCNTR enabled
NIDEN	SPNIDEN						
0	-	-	-	-	b0	No	Yes
0	-	-	-	-	b1	No	No
1	-	-	No	-	-	Yes	Yes
1	1	-	Yes	-	-	Yes	Yes
1	0	-	Yes	No	b0	No	Yes
1	0	-	Yes	No	b1	No	No

Table 10-12 Signal settings for the PMXEVCNTR (continued)

DBGEN	SPIDEN	SDER.SUNIDEN	Secure state	User mode	PMCR[5]	Performance counters enabled	PMCCNTR enabled
NIDEN	SPNIDEN						
1	0	0	Yes	Yes	b0	No	Yes
1	0	0	Yes	Yes	b1	No	No
1	0	1	Yes	Yes	-	Yes	Yes

10.2.12 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 modes 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 4-17 on page 4-18.

Figure 10-9 shows the PMUSERENR bit assignments.

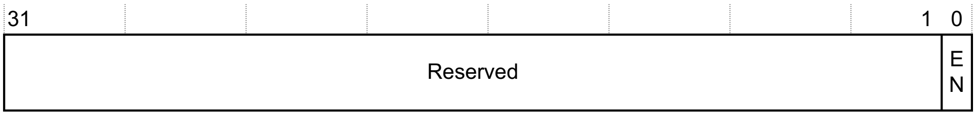


Figure 10-9 PMUSERENR bit assignments

Table 10-13 shows the PMUSERENR bit assignments.

Table 10-13 PMUSERENR bit assignments

Bits	Name	Description
[31:1]	Reserved	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
MCR p15, 0,<Rd>, c9, c14, 0 ; Write PMUSERENR
```

10.2.13 Interrupt Enable Set Register

The PMINTENSET characteristics are:

Purpose Determines if the PMCR or PMCCNTR generate an interrupt request on overflow. Interrupt requests are signaled by the Cortex-A5 processor using the **PMUIRQ** signal. See *Performance monitoring signals* on page A-8.

Usage constraints The PMINTENSET Register 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 4-17 on page 4-18.

Figure 10-10 shows the PMINTENSET Register bit assignments.

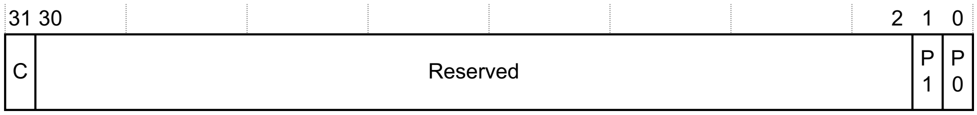


Figure 10-10 PMINTENSET Register bit assignments

Table 10-14 shows the PMINTENSET Register bit assignments.

Table 10-14 PMINTENSET Register bit assignments

Bits	Name	Description
[31]	C	PMCCNTR overflow interrupt request enable. When reading this register: 0 = interrupt request disabled 1 = interrupt request enabled. When writing to this register: 0 = no action 1 = interrupt request enabled.
[30:2]	Reserved	RAZ/WI.
[1]	P1	PMC1 overflow interrupt request enable.
[0]	P0	PMC0 overflow interrupt request 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 INTENC Register. See *Interrupt Enable Clear Register*.

10.2.14 Interrupt Enable Clear Register

The PMINTENCLR Register characteristics are:

Purpose Disables the generation of interrupt requests on overflows from the PMCR or PMCCNTR.

Usage constraints The PMINTENCLR Register 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 4-17 on page 4-18.

Figure 10-11 shows the PMINTENCLR Register bit assignments.

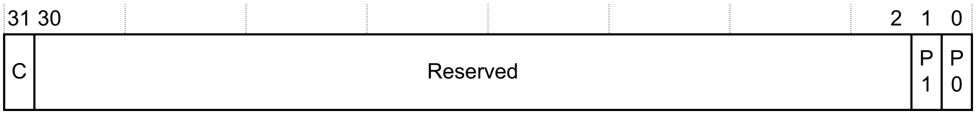


Figure 10-11 PMINTENCLR Register bit assignments

Table 10-15 shows the PMINTENCLR Register bit assignments.

Table 10-15 PMINTENCLR Register bit assignments

Bits	Name	Description
[31]	C	PMCCNTR overflow interrupt clear bit. When reading this register: 0 = interrupt request disabled 1 = interrupt request enabled. When writing to this register: 0 = no action 1 = interrupt request cleared.
[30:2]	Reserved	RAZ/WI.
[1]	P1	Clear interrupt request on PMC1 overflow.
[0]	P0	Clear interrupt request on PMC0 overflow.

To access the PMINTENCLR Register, use:

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

10.2.15 Configuration Register

The PMCFGR characteristics are:

Purpose Contains performance monitor specific configuration data.

Usage constraints There are no usage constraints

Configurations Available in all configurations

Attributes See the register summary in Table 10-1 on page 10-2.

Figure 10-12 on page 10-18 shows the PMCFGR bit assignments.

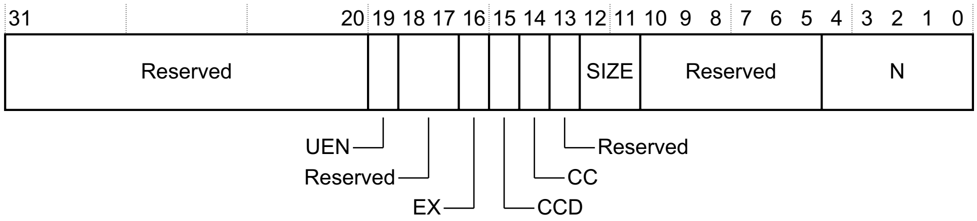


Figure 10-12 PMCFGR bit assignments

Table 10-16 shows the PMCFGR bit assignments.

Table 10-16 PMCFGR bit assignments

Bits	Name	Value	Description
[31:20]	-	RAZ	Reserved
[19]	UEN	0b1	User mode enable supported (using PMUSERENR)
[18:17]	-	RAZ	Reserved
[16]	EX	0b1	Event export supported
[15]	CCD	0b1	Cycle counter pre-scale supported
[14]	CC	0b1	Cycle counter implemented
[13]	-	RAZ	Reserved
[12:11]	SIZE	0b11	32-bit counters implemented
[10:8]	-	RAO	Reserved
[7:5]	-	RAZ	Reserved
[4:0]	N	`0x02`	2 event counters implemented

10.2.16 Lock Access Register

The PMLAR characteristics are:

Purpose Controls writes to the performance monitor registers. This reduces the risk of accidental corruption to the contents of the performance monitor registers. It does not prevent all accidental or malicious damage. Because the state of the register is in the debug power domain, it is not lost when the core powers down.

Usage constraints There are no usage constraints.

Configurations Available in all configurations.

Attributes See the register summary in Table 10-1 on page 10-2.

Figure 10-13 shows the PMLAR bit assignments.

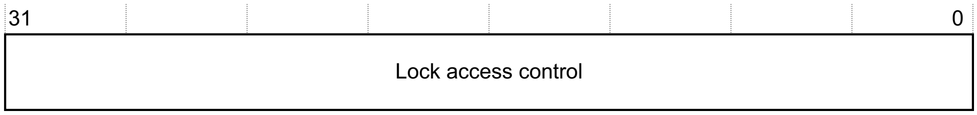


Figure 10-13 PMLAR bit assignments

Table 10-17 shows the PMLAR bit assignments.

Table 10-17 PMLAR bit assignments

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

When this register is written by an external debugger, APB read with **PADDRDBG31**=1, the results are Unpredictable.

10.2.17 Lock Status Register

The PMLSR characteristics are:

Purpose Returns the current lock status of the performance monitor registers.

Usage constraints There are no usage constraints.

Configurations Available in all configurations.

Attributes See the register summary in Table 10-1 on page 10-2.

Figure 10-14 shows the PMLSR bit assignments.

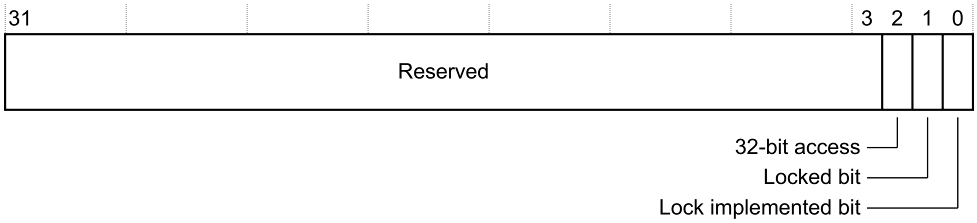


Figure 10-14 PMLSR bit assignments

Table 10-18 shows the PMLSR bit assignments.

Table 10-18 PMLSR bit assignments

Bits	Name	Description
[31:3]	Reserved	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 performance monitor registers lock. 0 = Lock clear. Performance monitor register writes are permitted. 1 = Lock set. Performance monitor 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`.

10.2.18 Authentication Status Register

The PMAUTHSTATUS Register characteristics are:

Purpose Reads the current values of the configuration inputs that determine the performance monitor permission level.

Usage constraints There are no usage constraints.

Configurations Available in all configurations.

Attributes See the register summary in Table 10-1 on page 10-2.

Figure 10-15 shows the PMAUTHSTATUS Register bit assignments.

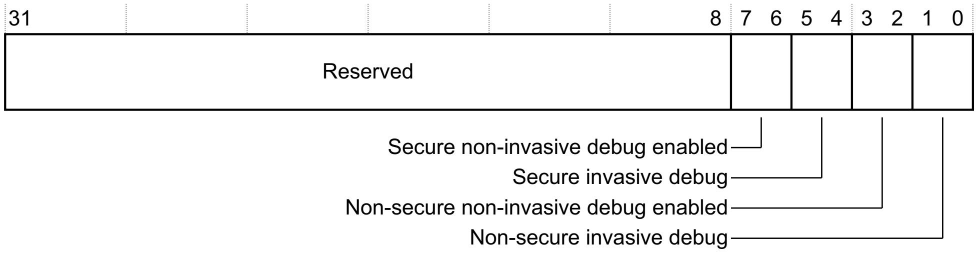

31
8
7
6
5
4
3
2
1
0
Reserved
Secure non-invasive debug enabled
Secure invasive debug
Non-secure non-invasive debug enabled
Non-secure invasive debug

Figure 10-15 PMAUTHSTATUS Register bit assignments

Table 10-19 shows the PMAUTHSTATUS bit assignments.

Table 10-19 PMAUTHSTATUS Register bit assignments

Bits	Name	Value	Description
[31:8]	Reserved	-	RAZ
[7]	Secure noninvasive debug enabled	b1	Secure noninvasive debug enable field
[6]		(**DBGEN** || **NIDEN**) && (**SPIDEN** || **SPNIDEN**)	
[5:4]	Secure invasive debug	RAZ	Secure invasive debug features not implemented
[3]	Non-secure noninvasive debug enabled	b1	Non-secure noninvasive debug enable field
[2]		**DBGEN** || **NIDEN**	
[1:0]	Non-secure invasive debug	RAZ	Non-secure invasive debug features not implemented

10.2.19 Device Type Register

The PMDEVTYPE Register characteristics are:

Purpose Indicates the type of performance monitor component.

Usage constraints There are no usage constraints.

Configurations Available in all configurations.

Attributes See the register summary in Table 10-1 on page 10-2.

Figure 10-16 on page 10-21 shows the PMDEVTYPE Register bit assignments.

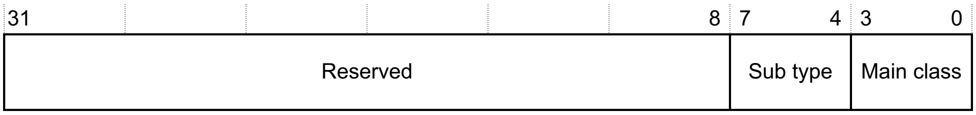


Figure 10-16 PMDEVTYPE Register bit assignments

Table 10-20 shows the PMDEVTYPE Register bit assignments.

Table 10-20 PMDEVTYPE Register bit assignments

Bits	Name	Description
[31:8]	Reserved	RAZ.
[7:4]	Sub type	Indicates that the sub-type of the Cortex-A5 processor is *core*. This value is `0x1`.
[3:0]	Main class	Indicates that the main class of the Cortex-A5 processor is *performance monitor*. This value is `0x6`.

10.2.20 Identification Registers

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

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

Table 10-21 shows the offset value, register number, and description that are associated with each Peripheral Identification Register.

Table 10-21 Peripheral Identification Registers

Offset (hex)	Register number	Description
`0x1FD0`	1012	Peripheral Identification Register 4
`0x1FD4`	1013	Reserved
`0x1FD8`	1014	Reserved
`0x1FDC`	1015	Reserved
`0x1FE0`	1016	Peripheral Identification Register 0
`0x1FE4`	1017	Peripheral Identification Register 1
`0x1FE8`	1018	Peripheral Identification Register 2
`0x1FEC`	1019	Peripheral Identification Register 3

Table 10-22 shows the Peripheral ID Register 0 bit assignments.

Table 10-22 Peripheral ID Register 0 bit assignments

Bits	Description
[31:8]	RAZ.
[7:0]	Indicates bits [7:0] of the part number for the Cortex-A5 processor. This value is `0xA5`.

Table 10-23 shows the Peripheral ID Register 1 bit assignments.

Table 10-23 Peripheral ID Register 1 bit assignments

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-A5 processor. This value is `0x9`.

Table 10-24 shows the Peripheral ID Register 2 bit assignments.

Table 10-24 Peripheral ID Register 2 bit assignments

Bits	Description
[31:8]	RAZ.
[7:4]	Indicates the revision number for the Cortex-A5 processor. This value changes based on the product major and minor revision. This value is set to 1 indicating revision r0p1.
[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 10-25 shows the Peripheral ID Register 3 bit assignments.

Table 10-25 Peripheral ID Register 3 bit assignments

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-A5 processor, this value is set to 0.

Table 10-26 shows the Peripheral ID Register 4 bit assignments.

Table 10-26 Peripheral ID Register 4 bit assignments

Bits	Description
[31:8]	RAZ.
[7:4]	Indicates the number of blocks occupied by the Cortex-A5 processor. This field is always set to 0.
[3:0]	Indicates the JEDEC JEP106 Continuation Code. For the Cortex-A5 processor, this value is `0x4`.

Table 10-27 shows the offset value, register number, and value that are associated with each Component Identification Register.

Table 10-27 Component Identification Registers

Offset (hex)	Register number	Value	Description
`0x1FF0`	1020	`0x0D`	Component Identification Register 0
`0x1FF4`	1021	`0x90`	Component Identification Register 1
`0x1FF8`	1022	`0x05`	Component Identification Register 2
`0x1FFC`	1023	`0xB1`	Component Identification Register 3

Appendix A
Signal Descriptions

This appendix describes the Cortex-A5 signals. It contains the following section:

- *Signal descriptions* on page A-2.

A.1 Signal descriptions

The following sections describe the Cortex-A5 signals:

- *Clock and reset signals*
- *Interrupt signals*
- *Configuration signals* on page A-3
- *Standby and wait for event signals* on page A-3
- *Power management signals* on page A-4
- *AXI interfaces* on page A-4
- *Performance monitoring signals* on page A-8
- *MBIST interface* on page A-8
- *Scan test signals* on page A-9
- *External debug interface* on page A-9
- *Trace interface signals* on page A-11.

A.1.1 Clock and reset signals

Table A-1 shows the clock and reset signals.

Table A-1 Clock and reset signals

Name	Type	Description
CLKIN	Input	Global clock.
nCPURESET	Input	Processor reset.
nDBGRESET	Input	Processor debug logic reset.
L1RSTDISABLE	Input	Disable invalidate entire data cache, instruction cache and TLB at reset.

See *Clocking and resets* on page 2-8.

A.1.2 Interrupt signals

Table A-2 shows the interrupt signals.

Table A-2 Interrupt signals

Name	Type	Description
nFIQ	Input	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	Input	Active-LOW interrupt request: 0 = activate interrupt 1 = do not activate interrupt. The processor treats the **nIRQ** input as level sensitive.

A.1.3 Configuration signals

Table A-3 shows the configuration signals.

Table A-3 Configuration signals

Name	Type	Description
CFGEND	Input	Controls the state of EE bit in the SCTLR: 0 = EE bit is LOW 1 = EE bit is HIGH This signal is only sampled during reset of the processor.
CLUSTERID[3:0]	Input	Value read in CPU ID field, bits[11:8], of the *Multiprocessor Affinity Register* (MPIDR).
CP15SDISABLE	Input	Disables write access to some system control processor registers: 0 = not enabled 1 = enabled. See Table 4-1 on page 4-4.
TEINIT	Input	Default exception handling state: 0 = ARM 1 = Thumb. It sets the SCTLR.TE bit. This signal is only sampled during reset of the processor.
VINITHI	Input	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 signal is only sampled during reset of the processor.

A.1.4 Standby and wait for event signals

Table A-4 shows the standby and wait for event signals.

Table A-4 Standby and wait for event signals

Name	Type	Description
EVENTI	Input	Event input for processor wake-up from WFE state.
EVENTO	Output	Event output. This signal is active when the `SEV` instruction is executed.
STANDBYWFI	Output	Indicates if the processor is in WFI mode: 0 = processor not in standby mode 1 = processor in standby mode. This is a pulse signal. An external master must latch the signal if it cannot deal with it immediately. See *Power control* on page 2-9 for information on how this signal is used.
STANDBYWFE	Output	Indicates if the processor is in WFE mode: 0 = processor not in wait for event mode 1 = processor in wait for event mode. This is a pulse signal. An external master must latch the signal if it cannot deal with it immediately. See *Power control* on page 2-9 for information on how this signal is used.

See *Communication to the Power Management Controller* on page 2-12.

A.1.5 Power management signals

Table A-5 shows the power management signals.

Table A-5 Power management signals

Name	Type	Description
CPURAMCLAMP	Input	Enable the RAM interface clamps, if implemented: 0 = clamps not active 1 = clamps active.

See *Communication to the Power Management Controller* on page 2-12.

A.1.6 AXI interfaces

The following sections describe the AXI interface signals:

- *Write address channel signals*
- *Write data channel signals* on page A-6
- *Write data response channel signals* on page A-6
- *Read address channel signals* on page A-6
- *Read data channel signals* on page A-7
- *AXI clock enable signal* on page A-8.

Write address channel signals

Table A-6 shows the AXI write address channel signals.

Table A-6 AXI write address channel signals

Name	Type	Description
AWADDR[31:0]	Output	Address.

AWBURST[1:0]	Output	Burst type:
		b01 = INCR incrementing burst
		b10 = WRAP wrapping burst.
		All other values are reserved.

AWCACHE[3:0]	Output	Cache type giving additional information about outer cacheable characteristics:
		b0000 = Strongly-ordered
		b0001 = Device
		b0011 = Normal, Non-Cacheable
		b0110 = Normal, Write-Through, no Write-Allocate
		b0111 = Normal, Write-Back, no Write-Allocate
		b1111 = Normal, Write-Back, Write-Allocate,

AWID[1:0]	Output	Request ID.

Table A-6 AXI write address channel signals (continued)

Name	Type	Description

AWLEN[3:0]	Output	Number of data transfers that can occur in 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
		.
		.
		.
		b1111 = 16 data transfers.

AWLOCK[1:0]	Output	Lock type:
		b00 = normal access
		b01 = exclusive access
		b1x = not used.

AWPROT[2:0]	Output	Protection type.
AWREADY	Input	Address ready.

AWSIZE[2:0]	Output	Burst size:
		b000 = 8-bit transfer
		b001 = 16-bit transfer
		b010 = 32-bit transfer
		b011 = 64-bit transfer.

AWUSER[6:0]	Output	[6:5] Exclusive mode:
		b00 = Not an eviction
		b01 = An eviction with dirty data
		b10 = Not used
		b11 = An eviction but the data is clean
		[4:0] Inner attributes:
		b00001 = Strongly ordered
		b00010 = Device, non-shareable
		b00011 = Device, shareable
		b00110 = Non-cacheable, non-shareable
		b00111 = Non-cacheable, shareable
		b11110 = Writeback cacheable, read and write allocate, non-shareable
		b11111 = Writeback cacheable, read and write allocate, shareable

AWVALID	Output	Address valid.

Write data channel signals

Table A-7 shows the AXI write data channel signals.

Table A-7 AXI write data channel signals

Name	Type	Description
WDATA[63:0]	Output	Write data.
WID[1:0]	Output	Write ID.
WLAST	Output	Write last indication.
WREADY	Input	Write ready.
WSTRB[7:0]	Output	Write byte lane strobe.
WVALID	Output	Write valid.

Write data response channel signals

Table A-8 shows the AXI write data response channel signals.

Table A-8 AXI write data response channel signals

Name	Type	Description
BID[1:0]	Input	Response ID.
BREADY	Output	Response ready.
BRESP[1:0]	Input	Write response.
BVALID	Input	Response valid.

Read address channel signals

Table A-9 shows the AXI read address channel signals.

Table A-9 AXI read address channel signals

Name	Type	Description
ARADDR[31:0]	Output	Address.

ARBURST[1:0]	Output	Burst type:
		b01 = INCR incrementing burst
		b10 = WRAP wrapping burst.
		All other values are reserved.

ARCACHE[3:0]	Output	Cache type giving additional information about outer cacheable characteristics:
		b0000 = Strongly-ordered
		b0001 = Device
		b0011 = Normal, Non-Cacheable
		b0110 = Normal, Write-Through, no Write-Allocate
		b0111 = Normal, Write-Back, no Write-Allocate
		b1111 = Normal, Write-Back, Write-Allocate.

ARID[2:0]	Output	Request ID.

Table A-9 AXI read address channel signals (continued)

Name	Type	Description

ARLEN[3:0]	Output	Number of data transfers that can occur in 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
		.
		.
		.
		b1111 = 16 data transfers.

ARLOCK[1:0]	Output	Lock type:
		b00 = normal access
		b01 = exclusive access
		b10 = locked access.

ARPROT[2:0]	Output	Protection type.
ARREADY	Input	Address ready.

ARSIZE[2:0]	Output	Burst size:
		b000 = 8-bit transfer
		b001 = 16-bit transfer
		b010 = 32-bit transfer
		b011 = 64-bit transfer.

ARUSER[4:0]	Output	Inner attributes:
		b00001 = Strongly ordered
		b00010 = Device, non-shareable
		b00011 = Device, shareable
		b00110 = Non-cacheable, non-shareable
		b00111 = Non-cacheable, shareable
		b11110 = Writeback cacheable, read and write allocate, non-shareable
		b11111 = Writeback cacheable, read and write allocate, shareable

ARVALID	Output	Address valid.

Read data channel signals

Table A-10 shows the AXI read data signals.

Table A-10 AXI read data signals

Name	Type	Description
RVALID	Input	Read valid.
RDATA[63:0]	Input	Read data.
RRESP[1:0]	Input	Read response.

Table A-10 AXI read data signals (continued)

Name	Type	Description
RLAST	Input	Read last.
RID[2:0]	Input	Read ID.
RREADY	Output	Read ready.

AXI clock enable signal

Table A-11 shows the AXI clock enable signal.

Table A-11 AXI clock enable signal

Name	Type	Description
ACLKEN	Input	Master AXI bus clock enable.

See *Clocking* on page 2-8.

A.1.7 Performance monitoring signals

Table A-12 shows the performance monitoring signal.

Table A-12 Performance monitoring signals

Name	Type	Description
PMUIRQ	Output	Performance Monitoring Unit interrupt signal.

A.1.8 MBIST interface

Table A-13 shows the MBIST interface signals.

Table A-13 MBIST interface signals

Name	Type	Description
MBISTADDR[11:0]	Input	MBIST address bus.
MBISTARRAY[5:0}	Input	MBIST arrays used for testing RAMs.
MBISTREQ	Input	MBIST test request.
MBISTWRITEEN	Input	Global write enable.
MBISTREADEN	Input	Global read enable.
MBISTBE[7:0]	Input	MBIST fine-grain write enable.
MBISTINDATA[71:0]	Input	MBIST data in.
MBISTACK	Output	MBIST test acknowledge.
MBISTOUTDATA[71:0]	Output	MBIST data out.

A.1.9 Scan test signals

Table A-14 shows the scan test signals.

Table A-14 Scan test signals

Signal	Type	Description
DFTSE	Input	Scan enable: 0 = not enabled 1 = enabled.
DFTRSTDISABLE	Input	Disable pipelined reset.

A.1.10 External debug interface

The following sections describe the external debug interface signals:

- *Authentication interface signals*
- *APB interface signals* on page A-10
- *CTI signals* on page A-10
- *Miscellaneous debug interface signals* on page A-11.

Authentication interface signals

Table A-15 shows the authentication interface signals.

Table A-15 Authentication interface signals

Name	Type	Description
DBGEN	Input	Invasive debug enable: 0 = not enabled 1 = enabled.
NIDEN	Input	Noninvasive debug enable: 0 = not enabled 1 = enabled.
SPIDEN	Input	Secure privileged invasive debug enable: 0 = not enabled 1 = enabled.
SPNIDEN	Input	Secure privileged noninvasive debug enable: 0 = not enabled 1 = enabled.

APB interface signals

Table A-16 shows the APB interface signals.

Table A-16 APB interface signals

Name	Type	Description
PADDRDBG[12:2]	Input	Programming address. Use **PADDRDBG[12]** = b0 to access the debug registers, see Chapter 9 *Debug*. Use **PADDRDBG[12]** = b1 to access the Performance Monitoring registers, see Chapter 10 *Performance Monitoring Unit*.
PADDRDBG31	Input	APB address bus bit [31]: 0 = not an external debugger access 1 = external debugger access.
PCLKENDBG	Input	Clock enable for debug APB interface.
PENABLEDBG	Input	Second and subsequent cycle of transfer.
PRDATADBG[31:0]	Output	APB read data bus.
PREADYDBG	Output	APB slave ready. An APB slave can assert **PREADY** to extend a transfer.
PSELDBG	Input	Debug/performance monitoring registers select: 0 = registers not selected 1 = registers selected.
PSLVERRDBG	Output	APB slave error signal.
PWDATADBG[31:0]	Input	APB write data.
PWRITEDBG	Input	APB read/write signal.

CTI signals

Table A-17 shows the CTI signals.

Table A-17 CTI signals

Name	Type	Description
DBGACK	Output	Debug acknowledge signal.
DBGRESTART	Input	Causes the processor to exit from Debug state. It must be held HIGH until **DBGRESTARTED** is deasserted. 0 = not enabled 1 = enabled.
DBGRESTARTED	Output	Used with **DBGRESTART** to move between Debug state and Normal state. 0 = not enabled 1 = enabled.
DBGTRIGGER	Output	Indicates that the processor is committed to entering debug state. Active HIGH.
EDBGRQ	Input	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**.

Miscellaneous debug interface signals

Table A-18 shows the miscellaneous debug interface signals.

Table A-18 Miscellaneous debug signals

Name	Type	Description
COMMRX	Output	Communications channel receive. Receive portion of Data Transfer Register full flag: 0 = empty 1 = full.
COMMTX	Output	Communications channel transmit. Transmit portion of Data Transfer Register empty flag: 0 = full 1 = empty.
DBGNOPWRDWN	Output	Debugger has requested the processor is not powered down.
DBGSWENABLE	Input	When HIGH only the external debug agent can modify debug registers. 0 = not enabled. This is the default. 1 = enabled.
DBGROMADDR[31:12]	Input	Specifies bits [31:12] of the ROM table physical address. If the address cannot be determined tie this signal off to zero.
DBGROMADDRV	Input	Valid signal for **DBGROMADDR**. If the address cannot be determined tie this signal LOW.
DBGSELFADDR[31:12]	Input	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	Input	Valid signal for **DBGSELFADDR**. If the offset cannot be determined tie this signal LOW.

See Chapter 9 *Debug*.

A.1.11 Trace interface signals

Table A-19 shows the trace interface signals. In the Type column:

- Input indicates an input from the trace interface to the processor
- Output indicates an output from the processor to the trace interface.

All these signals are in the processor clock domain, **CLKIN**.

Table A-19 Trace interface signals

Name	Type	Description
ETMICTL[19:0]	Output	ETM instruction control bus.
ETMIA[31:0]	Output	ETM instruction address.
ETMDCTL[10:0]	Output	ETM data control bus.
ETMDA[31:0]	Output	ETM data address.
ETMDD[31:0]	Output	ETM data write data value.

Table A-19 Trace interface signals (continued)

Name	Type	Description
ETMCID[31:0]	Output	Current processor Context ID.
ETMWFXPENDING	Output	Processor is attempting to enter WFI state.
EVNTBUS[29:0]	Output	Performance monitor unit output.
ETMPWUP	Input	Power up processor ETM interface.
ETMEXTOUT[1:0]	Input	ETM external event to be monitored.

See *ETM interface* on page 2-7.

Appendix B
Revisions

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

Table B-1 Issue A

Change	Location	Affects
No changes, first release	-	-

Table B-2 Differences between issue A and issue B

Change	Location	Affects
Invalidate TLB entry by VA added	*Invalidate TLB entry by VA* on page 4-10	All
Invalidate TLB entries by VA all ASID description updated	*Invalidate TLB entries by VA all ASID* on page 4-10	All
Main ID Register reset value updated	Table 4-9 on page 4-14 Table 9-16 on page 9-29	r0p1
Reset values updated for: • Cache Type Register • Multiprocessor Affinity Register • Memory Model Feature Register 3	Table 4-9 on page 4-14 Table 9-16 on page 9-29	All
CTR bit descriptions updated	Table 4-23 on page 4-22	All
BP maintain bit description updated	Table 4-32 on page 4-30	All
DWBST bit description updated	Table 4-44 on page 4-42	All

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

Change	Location	Affects
Configuration Base Address Register description updated	*Configuration Base Address Register* on page 4-66	All
Jazelle Identity and Miscellaneous Functions Register reset value updated	Table 5-1 on page 5-3	All
Jazelle Configurable Opcode Translation Table Register access updated	*Jazelle Configurable Opcode Translation Table Register* on page 5-8	All
Supported AXI transfers updated	*Supported AXI transfers* on page 8-3	All
Operating System Lock and Save/Restore Register accesses updated	Table 9-1 on page 9-8	All
Reset values updated for: • TLB Type Register • Multiprocessor Affinity Register • Debug Feature Register 0 • Memory Model Feature Register 1 • Memory Model Feature Register 3	Table 9-16 on page 9-29	All
Peripheral ID Register 2 [7:4] description updated	Table 9-26 on page 9-35	r0p1
Common Event Identification Register 0 and 1 added	Table 10-1 on page 10-2	All
Peripheral ID Register 2 [7:4] description updated	Table 10-24 on page 10-22	r0p1
AWCACHE description updated	Table A-6 on page A-4	All
ARCACHE description updated	Table A-9 on page A-6	All
DFTRSTDISABLE moved to scan test signals	Table A-14 on page A-9	All

Table B-3 Differences betwwen issue B and issue C

Change	Location	Affects
ACTLR bit assignments updated	Table 4-44 on page 4-42	Iccue C
DBGPRCR bit assignments updated	Table 9-13 on page 9-27	Issue C

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.

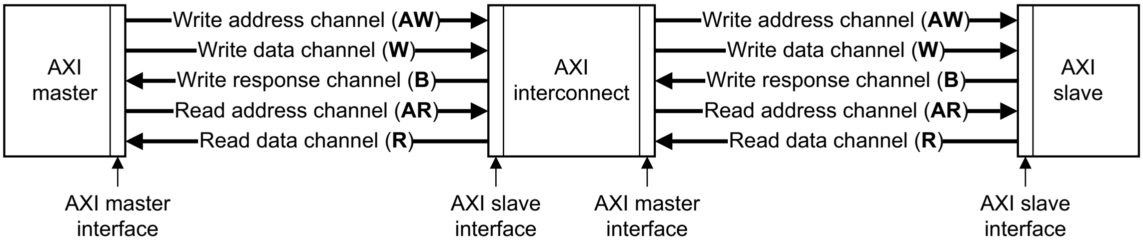
AXI

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).

[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.

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.

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.

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 ARMv7 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 privileged modes.

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.

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.

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. 754-2008*. 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.

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.

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.

PA

See Physical Address.

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 Lookaside 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 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.

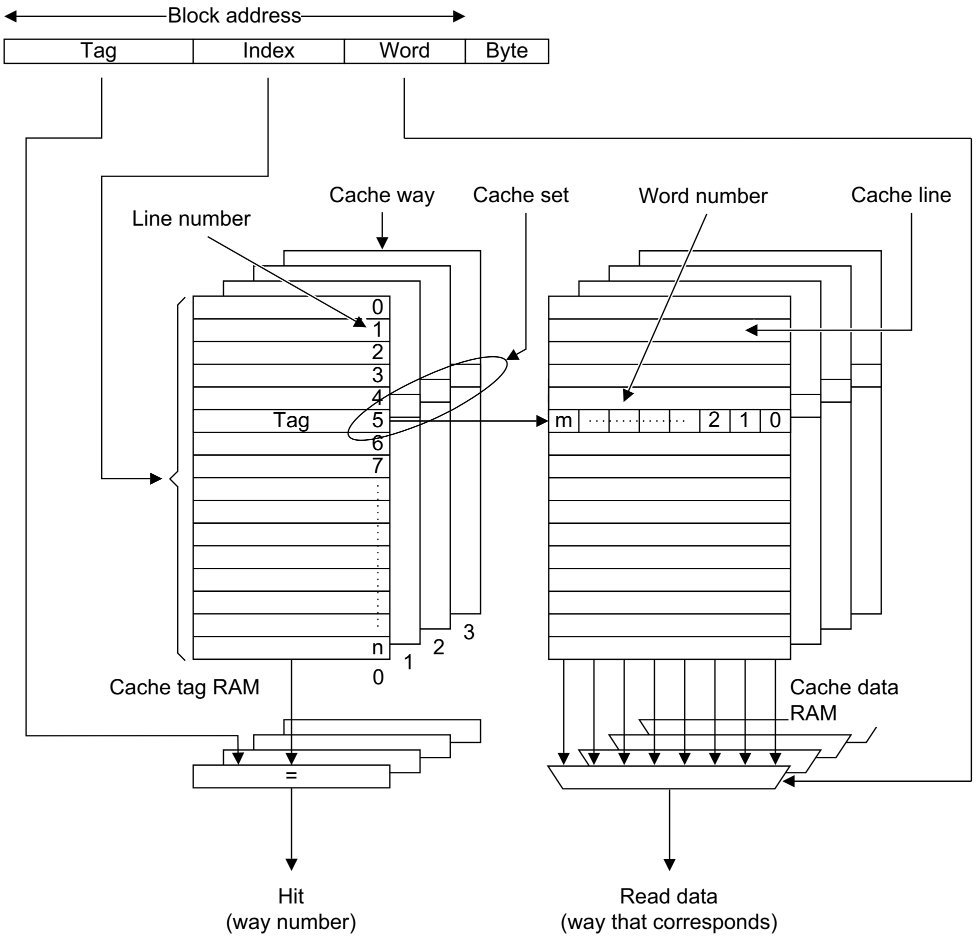
Write-through (WT) In a write-through cache, data is written to main memory at the same time as the cache is updated.

WT *See* Write-through.

Cache terminology diagram

The diagram illustrates the following cache terminology:

- block address
- cache line
- cache set
- cache way
- index
- tag.


Block address
Tag
Index
Word
Byte
Cache way
Cache set
Word number
Cache line
Line number
0
1
2
3
4
Tag
5
6
7
n
1
2
3
0
m
2
1
0
Cache tag RAM
Cache data RAM
=
Hit (way number)
Read data (way that corresponds)