Arm® Cortex®-A75 Core

Revision: r2p1

Technical Reference Manual

arm

100403_0201_00_en

Arm® Cortex®-A75 Core

Technical Reference Manual

Release Information

Document History

Issue	Date	Confidentiality	Change
0000-00	30 September 2016	Confidential	First release for r0p0
0100-00	12 December 2016	Confidential	First release for r1p0
0200-00	03 June 2017	Non-Confidential	First release for r2p0
0201-00	20 October 2017	Non-Confidential	First release for r2p1

Non-Confidential Proprietary Notice

Arm Limited. Company 02557590 registered in England.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

LES-PRE-20349

Confidentiality Status

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

Unrestricted Access is an Arm internal classification.

Product Status

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

Web Address

http://www.arm.com

Contents
Arm® Cortex®-A75 Core Technical Reference Manual

Part B Register descriptions

Chapter B1 AArch32 system registers

Chapter B2 AArch64 system registers

Preface

This preface introduces the *Arm® Cortex®-A75 Core Technical Reference Manual*.

It contains the following:

About this book

This Technical Reference Manual is for the Cortex®-A75 core. It provides reference documentation and contains programming details for registers. It also describes the memory system, the caches, the interrupts, and the debug features.

Product revision status

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

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

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

Intended audience

This manual is for system designers, system integrators, and programmers designing or programming a *System-on-Chip* (SoC) using an Arm core.

Using this book

This book is organized into the following chapters:

Part A Functional description

This part describes the main functionality of the Cortex-A75 core.

Chapter A1 Introduction

This chapter provides an overview of the Cortex-A75 core and its features.

Chapter A2 Technical overview

This chapter describes the structure of the Cortex-A75 core.

Chapter A3 Clocks, resets, and input synchronization

This chapter describes the clocks, resets, and input synchronization of the Cortex-A75 core.

Chapter A4 Power management

This chapter describes the power domains and the power modes in the Cortex-A75 core.

Chapter A5 Memory Management Unit

This chapter describes the *Memory Management Unit* (MMU) of the Cortex-A75 core.

Chapter A6 Level 1 memory system

This chapter describes the L1 instruction cache and data cache that make up the L1 memory system.

Chapter A7 Level 2 memory system

This chapter describes the L2 memory system.

Chapter A8 Reliability, Availability, and Serviceability (RAS)

This chapter describes the RAS features implemented in the Cortex-A75 core.

Chapter A9 Generic Interrupt Controller CPU Interface

This chapter describes the Cortex-A75 core implementation of the Arm *Generic Interrupt Controller* (GIC) CPU interface.

Chapter A10 Advanced SIMD and Floating-point support

This chapter describes the Advanced SIMD and floating-point features and registers in the Cortex-A75 core. The unit in charge of handling the Advanced SIMD and floating-point features is also referred to as data engine in this manual.

Part B Register descriptions

This part describes the non-debug registers of the Cortex-A75 core.

Chapter B1 AArch32 system registers
This chapter describes the system registers in the AArch32 state.

Chapter B2 AArch64 system registers
This chapter describes the system registers in the AArch64 state.

Chapter B3 Error system registers
This chapter describes the error registers accessed by both the AArch32 error registers and the AArch64 error registers.

Chapter B4 GIC registers
This chapter describes the GIC registers.

Chapter B5 Advanced SIMD and floating-point registers
This chapter describes the Advanced SIMD and Floating-point registers.

Part C Debug descriptions
This part describes the debug functionality of the Cortex-A75 core.

Chapter C1 Debug
This chapter describes the Cortex-A75 core debug registers and shows examples of how to use them.

Chapter C2 Performance Monitor Unit
This chapter describes the *Performance Monitor Unit* (PMU) and the registers that it uses.

Chapter C3 Activity Monitor Unit
This chapter describes the *Activity Monitor Unit* (AMU).

Chapter C4 Embedded Trace Macrocell
This chapter describes the ETM for the Cortex-A75 core.

Part D Debug registers
This part describes the debug registers of the Cortex-A75 core.

Chapter D1 AArch32 debug registers
This chapter describes the debug registers in the AArch32 Execution state and shows examples of how to use them.

Chapter D2 AArch64 debug registers
This chapter describes the debug registers in the AArch64 Execution state and shows examples of how to use them.

Chapter D3 Memory-mapped debug registers
This chapter describes the memory-mapped debug registers and shows examples of how to use them.

Chapter D4 AArch32 PMU registers
This chapter describes the AArch32 PMU registers and shows examples of how to use them.

Chapter D5 AArch64 PMU registers
This chapter describes the AArch64 PMU registers and shows examples of how to use them.

Chapter D6 Memory-mapped PMU registers
This chapter describes the memory-mapped PMU registers and shows examples of how to use them.

Chapter D7 AArch64 AMU registers
This chapter describes the AArch64 AMU registers and shows examples of how to use them.

Chapter D8 Memory-mapped AMU registers
This chapter describes the memory-mapped AMU registers and shows examples of how to use them.

Chapter D9 PMU snapshot registers

PMU snapshot registers are an implementation defined extension to an Armv8 compliant PMU to support an external core monitor that connects to a system profiler.

Chapter D10 ETM registers

This chapter describes the ETM registers.

Part E Appendices

This part describes the appendices of the Cortex-A75 core.

Appendix A Cortex®-A75 Core AArch32 unpredictable Behaviors

This appendix describes specific Cortex-A75 core UNPREDICTABLE behaviors of particular interest.

Appendix B Revisions

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

Glossary

The Arm® Glossary is a list of terms used in Arm documentation, together with definitions for those terms. The Arm Glossary does not contain terms that are industry standard unless the Arm meaning differs from the generally accepted meaning.

See the *Arm® Glossary* for more information.

Typographic conventions

italic

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

bold

Highlights interface elements, such as menu names. Denotes signal names. Also used for terms in descriptive lists, where appropriate.

`monospace`

Denotes text that you can enter at the keyboard, such as commands, file and program names, and source code.

`monospace`

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

`monospace italic`

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

`monospace bold`

Denotes language keywords when used outside example code.

`<and>`

Encloses replaceable terms for assembler syntax where they appear in code or code fragments. For example:

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

SMALL CAPITALS

Used in body text for a few terms that have specific technical meanings, that are defined in the *Arm® Glossary*. For example, IMPLEMENTATION DEFINED, IMPLEMENTATION SPECIFIC, UNKNOWN, and UNPREDICTABLE.

Timing diagrams

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

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

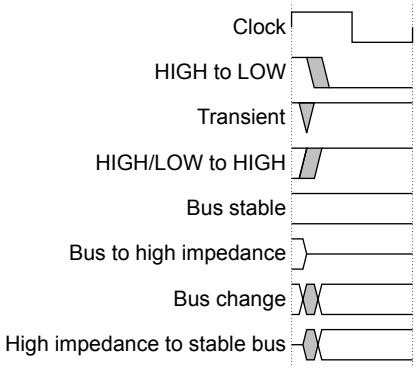

Figure 1 Key to timing diagram conventions

Signals

The signal conventions are:

Signal level

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

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

Lowercase n

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

Additional reading

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

Arm publications

- *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile* (DDI 0487).
- *AMBA® AXI™ and ACE™ Protocol Specification AXI3™, AXI4™, and AXI4-Lite™, ACE and ACE-Lite™* (IHI 0022).
- *Arm® AMBA® CHI Architecture Specification* (IHI 0050).
- *AMBA® APB Protocol Version 2.0 Specification* (IHI 0024).
- *AMBA® 4 ATB Protocol Specification* (IHI 0032).
- *Arm® CoreSight™ Architecture Specification v3.0* (IHI 0029).
- *Arm® Embedded Trace Macrocell Architecture Specification ETMv4* (IHI 0064).
- *Low Power Interface Specification Arm® Q-Channel and P-Channel Interfaces* (IHI 0068).
- *Arm® Debug Interface Architecture Specification, ADIv5.0 to ADIv5.2* (IHI 0031).
- *Arm® Generic Interrupt Controller Architecture Specification* (IHI 0069).
- *Arm® CoreSight™ ELA-500 Embedded Logic Analyzer* (100127).

The following confidential documents are only available to licensees:

- *Arm® Cortex®-A75 Core Cryptographic Extension Technical Reference Manual* (100458).
- *Arm® Cortex®-A75 Core Configuration and Sign-off Guide* (100404).
- *Arm® Cortex®-A75 Core Integration Manual* (100405).
- *Arm® DynamIQ™ Shared Unit Technical Reference Manual* (100453).

Other publications

- *ANSI/IEEE Std 754-2008, IEEE Standard for Binary Floating-Point Arithmetic*

  ———— **Note** ————

  Arm floating-point terminology is largely based on the earlier ANSI/IEEE Std 754-1985 issue of the standard. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile* for more information.

  ————————————

Feedback

Feedback on this product

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

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

Feedback on content

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

- The title *Arm Cortex-A75 Core Technical Reference Manual*.
- The number 100403_0201_00_en.
- If applicable, the page number(s) to which your comments refer.
- A concise explanation of your comments.

Arm also welcomes general suggestions for additions and improvements.

——— **Note** ———

Arm tests the PDF only in Adobe Acrobat and Acrobat Reader, and cannot guarantee the quality of the represented document when used with any other PDF reader.

———————————

Part A
Functional description

Chapter A1
Introduction

This chapter provides an overview of the Cortex-A75 core and its features.

It contains the following sections:

A1.1 About the core

The Cortex-A75 core is a high-performance and low-power Arm product that implements the Armv8-A architecture with support for the v8.2 extension, including the RAS extension, and the Load acquire (`LDAPR`) instructions introduced in the v8.3 extension.

In this manual, this feature is referred to as the `LDAPR` instructions.

In an Arm *DynamIQ Shared Unit* (DSU) cluster , each Cortex-A75 core has a *Level 1* (L1) memory system and a private, integrated *Level 2* (L2) cache. It also includes a superscalar, variable-length, out-of-order pipeline. Each Cortex-A75 core can handle a sustained maximum throughput of three instructions per cycle.

The following figure shows an example of a configuration with four Cortex-A75 cores.

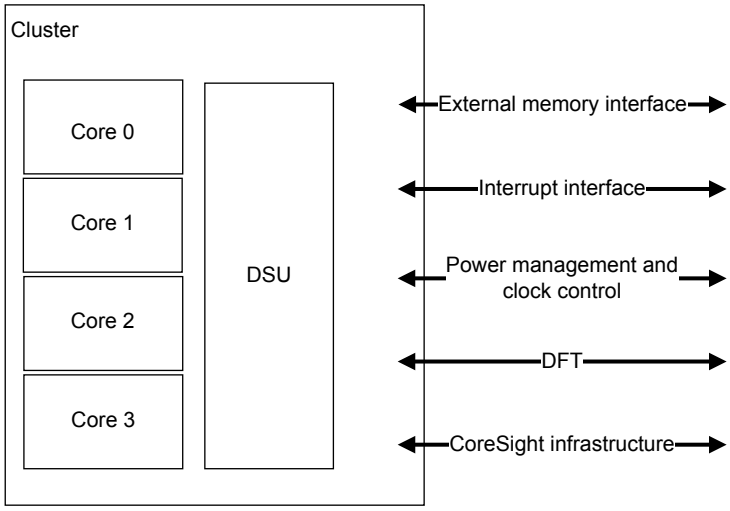

Figure A1-1 Example Cortex-A75 core configuration

For more information on the permissible combination of cores in the cluster, see appendix *Compatible Core Versions* in the *Arm®DynamIQ™ Shared Unit Technical Reference Manual.*

A1.2 Features

The Cortex-A75 core might be used in standalone DynamIQ configurations, that is in a homogenous cluster of one to four Cortex-A75 cores with no LITTLE cores.

It might also be used as the *big* core in a DynamIQ big.LITTLE arrangement, and is fully compatible with the big.LITTLE use models introduced by previous generations of Cortex products.

The Cortex-A75 core includes the following features:

Core features

- Full implementation of the Armv8.2-A A64, A32, and T32 instruction sets.
- Both the AArch64 and AArch32 execution states at all Exception levels (EL0 to EL3).
- TrustZone®.
- A *Memory Management Unit* (MMU).
- Variable-length, out-of-order pipeline with symmetrical three-way superscalar.
- 44-bit *Physical Address* (PA).
- An integrated Data Engine unit that implements the Advanced SIMD and floating-point architecture support.
- Optional Cryptographic Extension.
- *Generic Interrupt Controller* (GIC) CPU interface to connect to an external distributor.
- Generic Timers interface supporting 64-bit count input from an external system counter.

Cache features

- Separate L1 data and instruction caches.
- Private, unified data and instruction L2 cache.
- L1 and L2 memory protection in the form of *Error Correction Code* (ECC) or parity on all RAM instances.

Debug features

- RAS Extension.
- Armv8.2 debug logic.
- *Performance Monitoring Unit* (PMU).
- *Embedded Trace Macrocell* (ETM) that supports instruction trace only.

See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile* for more information.

A1.3 Implementation options

All the Cortex-A75 cores in the DSU must have the same build-time configuration.

The following table lists the implementation options for a Cortex-A75 core.

Table A1-1 Cortex-A75 core implementation options

Feature	Range of options
L2 cache size	• 256KB. • 512KB.
L1 and L2 error protection	Can be included or not included.
Cryptographic Extension	Can be included or not included.
L2/L3 interface	Can be synchronous or asynchronous.
Core ELA	Can be included or not included.

Note

- In a big.LITTLE system, the Cryptographic Extension must be included or not included in a homogeneous way in all cores within the DSU.
- Error protection consists of:
  - Parity on any RAM that contains only clean data and which could induce failure.
  - ECC-SECDED on any RAM that potentially contains dirty data.

  If RAM corruption only induces a performance degradation, then the RAMs are not protected against errors.
- L1 and L2 error protection can only be enabled if L3 error protection is enabled.

A1.4 Supported standards and specifications

The Cortex-A75 core implements the Armv8-A architecture and some architecture extensions. It also supports interconnect, interrupt, timer, debug, and trace architectures.

Table A1-2 Compliance with standards and specifications

Architecture specification or standard	Version	Notes
Arm architecture	Armv8-A	• AArch64 and AArch32 execution states at all Exception levels. • A64, A32, and T32 instruction sets.
Arm architecture extensions	• v8.1 extensions. • v8.2 extensions. • Cryptographic extension. • RAS extension. • v8.3 extensions. • v8.4 extensions.	• From the v8.2 extensions, the Cortex-A75 core does not implement the optional Statistical Profiling Extension. • From the v8.3 extensions, the Cortex-A75 core only implements the `LDAPR` instructions. • From the v8.4 extensions, the Cortex-A75 core only implements the `UDOT` and `SDOT`instructions.
Generic Interrupt Controller	GICv4	-
Generic Timer	Armv8-A	64-bit external system counter with timers within each core.
PMU	PMUv3	-
Debug	Armv8-A	With support for the debug features added by the v8.2 extensions.
CoreSight	CoreSightv3	-
Embedded Trace Macrocell	ETMv4.2	Instruction trace only.

See *Additional reading* on page 21 for a list of architectural references.

A1.5 Test features

The Cortex-A75 core provides test signals that enable the use of both *Automatic Test Pattern Generation* (ATPG) and *Memory Built-In Self Test* (MBIST) to test the core logic and memory arrays.

A1.6 Design tasks

The Cortex-A75 core is delivered as a synthesizable *Register Transfer Level* (RTL) description in Verilog HDL. Before you can use the Cortex-A75 core, you must implement it, integrate it, and program it.

A different party can perform each of the following tasks. Each task can include implementation and integration choices that affect the behavior and features of the core.

Implementation
The implementer configures and synthesizes the RTL to produce a hard macrocell. This task includes integrating RAMs into the design.

Integration
The integrator connects the macrocell into a SoC. This task includes connecting it to a memory system and peripherals.

Programming
In the final task, the system programmer develops the software to configure and initialize the core and tests the application software.

The operation of the final device depends on the following:

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

Configuration inputs
The integrator configures some features of the core by tying inputs to specific values. These configuration settings 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 core by programming particular values into registers. The configuration choices affect the behavior of the core.

A1.7 Product revisions

This section indicates the first release and, in subsequent releases, describes the differences in functionality between product revisions.

r0p0

First release.

r1p0

There are minor changes that affect the programmers model. For more information, see *B.1 Revisions* on page Appx-B-796.

r2p0

This release includes a new activity monitoring feature and instructions from the v8.4 architecture extensions. For more information, see *B.1 Revisions* on page Appx-B-796.

——— **Note** ———

This feature must be used with the r1p0 revision of the DSU.

———————————

r2p1

This release includes documentation errata fixes. For more information, see *B.1 Revisions* on page Appx-B-796.

Chapter A2
Technical overview

This chapter describes the structure of the Cortex-A75 core.

It contains the following sections:

A2.1 Components

In a standalone configuration, there can be up to four Cortex-A75 cores and a DSU that connects the cores to an external memory system.

For more information about the DSU, see the *Arm® DynamIQ™ Shared Unit Technical Reference Manual*.

The main components of the Cortex-A75 core are:

- Instruction fetch.
- Instruction decode.
- Register renaming.
- Instruction dispatch.
- Execution pipelines.
- L1 data memory system.
- L2 memory system.

The following figure is an overview of the Cortex-A75 core.

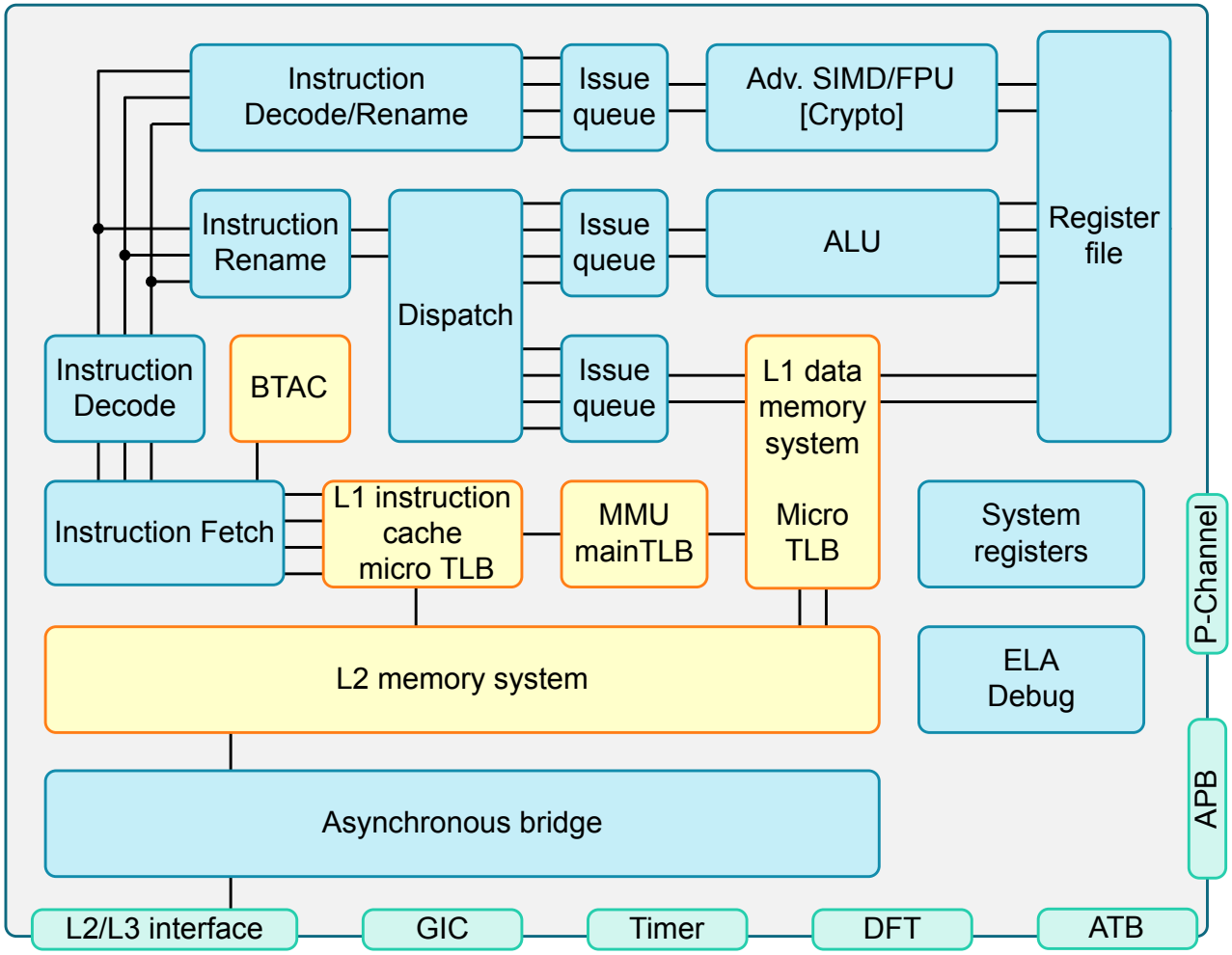

Figure A2-1 Cortex-A75 core overview

A2.1.1 Instruction fetch

The *Instruction Fetch Unit* (IFU) fetches instructions from the L1 instruction cache and delivers up to three instructions per cycle to the instruction decode unit.

The IFU includes:

- A 64KB, 4-way, set associative L1 instruction cache with 64-byte cache lines and optional dual-bit parity protection.
- A fully associative instruction micro TLB with native support for 4KB, 64KB, and 1MB page sizes.
- A 2-level dynamic branch predictor.

A2.1.2 Instruction decode

The instruction decode unit supports the A32, T32, and A64 instruction sets. It also supports Advanced SIMD and floating-point instructions in each instruction set.

A2.1.3 Register renaming

The Cortex-A75 core performs register renaming to facilitate out-of-order execution by removing *Write-After-write* (WAW) and *Write-After-read* (WAR) hazards.

A2.1.4 Instruction dispatch

The instruction dispatch unit controls when the decoded instructions are dispatched to the execution pipelines. It includes Issue Queues for storing instruction pending dispatch to execution pipelines.

A2.1.5 Execution pipeline

The execution pipeline includes:

- Two *Arithmetic Logical Unit* (ALU) pipelines including integer multiply-accumulate and iterative integer divide hardware.
- Branch and instruction condition codes resolution logic.
- Advanced SIMD and Floating Point pipelines, referred to in this manual as data engine. Optionally, the data engine can execute the cryptographic instructions.

A2.1.6 L1 data memory system

The L1 data memory system executes load and store instructions and encompasses the L1 data side memory system. It also services memory coherency requests.

The load/store unit includes:

- A 64KB, 16-way, set associative cache with 64-byte cache lines and optional ECC protection per 32 bits.
- A fully associative L1 data TLB with native support for 4KB, 64KB, and 1MB page sizes.

A2.1.7 L2 memory system

The L2 memory system services L1 instruction and data cache misses from the Cortex-A75 core.

The L2 memory system includes:

- An 8-way set associative L2 cache with data ECC protection per 64 bits. The L2 cache is configurable with sizes of 256KB or 512KB.
- An interface with the DSU configurable at implementation time for synchronous or asynchronous operation.

A2.2 Interfaces

The Cortex-A75 core has several interfaces to connect it to a SoC. The DSU manages all interfaces.

For information on the interfaces, see the *Arm® DynamIQ™ Shared Unit Technical Reference Manual*.

A2.3 About system control

The system registers control and provide status information for the functions that the core implements.

The main functions of the system registers are:

- Overall system control and configuration.
- MMU configuration and management.
- Cache configuration and management.
- System performance monitoring.
- GIC configuration and management.

The system registers are accessible in the AArch64 and AArch32 Execution states. Some of the system registers are accessible through the external debug interface.

A2.4 About the Generic Timer

The Generic Timer can schedule events and trigger interrupts that are based on an incrementing counter value. It generates timer events as active-LOW interrupt outputs and event streams.

The Cortex-A75 core provides a set of timer registers. The timers are:

- An EL1 Non-secure physical timer.
- An EL2 Hypervisor physical timer.
- An EL3 Secure physical timer.
- A virtual timer.
- A Hypervisor virtual timer.

The Cortex-A75 core does not include the system counter. This resides in the SoC. The system counter value is distributed to the core with a 64-bit bus.

For more information on the Generic Timer, see the *Arm® DynamIQ™ Shared Unit Technical Reference Manual* and the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

Chapter A3
Clocks, resets, and input synchronization

This chapter describes the clocks, resets, and input synchronization of the Cortex-A75 core.

It contains the following sections:

A3.1 About Clocks, Resets, and Input Synchronization

The Cortex-A75 core supports hierarchical clock gating.

The Cortex-A75 core contains several interfaces that connect to other components in the system. These interfaces can be in the same clock domain or in other clock domains.

For information about clocks, resets, and input synchronization, see the *Arm® DynamIQ™ Shared Unit Technical Reference Manual*.

A3.2 Asynchronous interface

Your implementation can include an optional asynchronous interface between the core and the DSU top level.

See the *Arm® DynamIQ™ Shared Unit Technical Reference Manual* for more information.

Chapter A4
Power management

This chapter describes the power domains and the power modes in the Cortex-A75 core.

It contains the following sections:

A4.1 About power management

The Cortex-A75 core provides mechanisms to control both dynamic and static power dissipation.

Dynamic power management includes the following features:

- Architectural clock gating.
- Per-core *Dynamic Voltage and Frequency Scaling* (DVFS).

Static power management includes the following features:

- Dynamic retention.
- Powerdown.

A4.2 Voltage domains

The Cortex-A75 core supports a VCPU voltage domain and a VSYS voltage domain.

The following figure shows the VCPU and VSYS voltage domains in each Cortex-A75 core and in the DSU. The example shows a configuration with four Cortex-A75 cores.

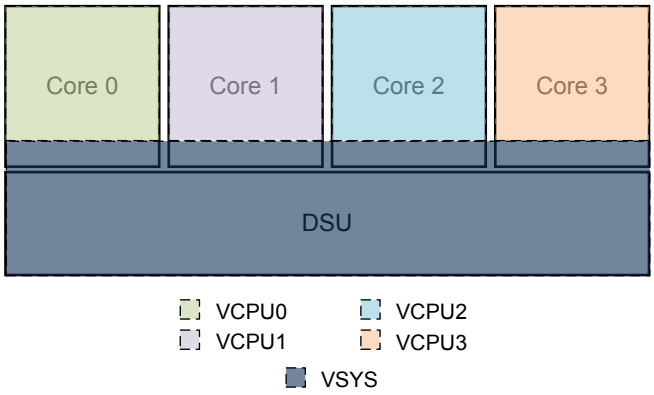

Figure A4-1 Cortex-A75 Voltage Domains

Asynchronous bridge logic exists between the voltage domains. The Cortex-A75 core logic and core clock domain of the asynchronous bridge are in the VCPU voltage domain. The DSU clock domain of the asynchronous bridge is in the VSYS voltage domain.

Note

You can tie VCPU and VSYS to the same supply if the system does not require per-core DVFS.

A4.3 Power domains

The DSU can support up to four `prometheus_core.v` units.

Each `prometheus_core.v` unit supports two power domains:

- A PDCPU power domain that contains all prometheus_cpu logic and part of the cpu asynchronous bridge.
  - The Advanced SIMD and floating-point unit is included in the PDCPU power domain and is not supported as a separate power domain.
  - The L1 and L2 RAMs are included in the PDCPU power domain and are not part of a separate power domain.
- A PDSYS power domain that contains the part of the cpu asynchronous bridge that belongs to the DSU power domain.

The following figure shows an example of the organization of the power domains.

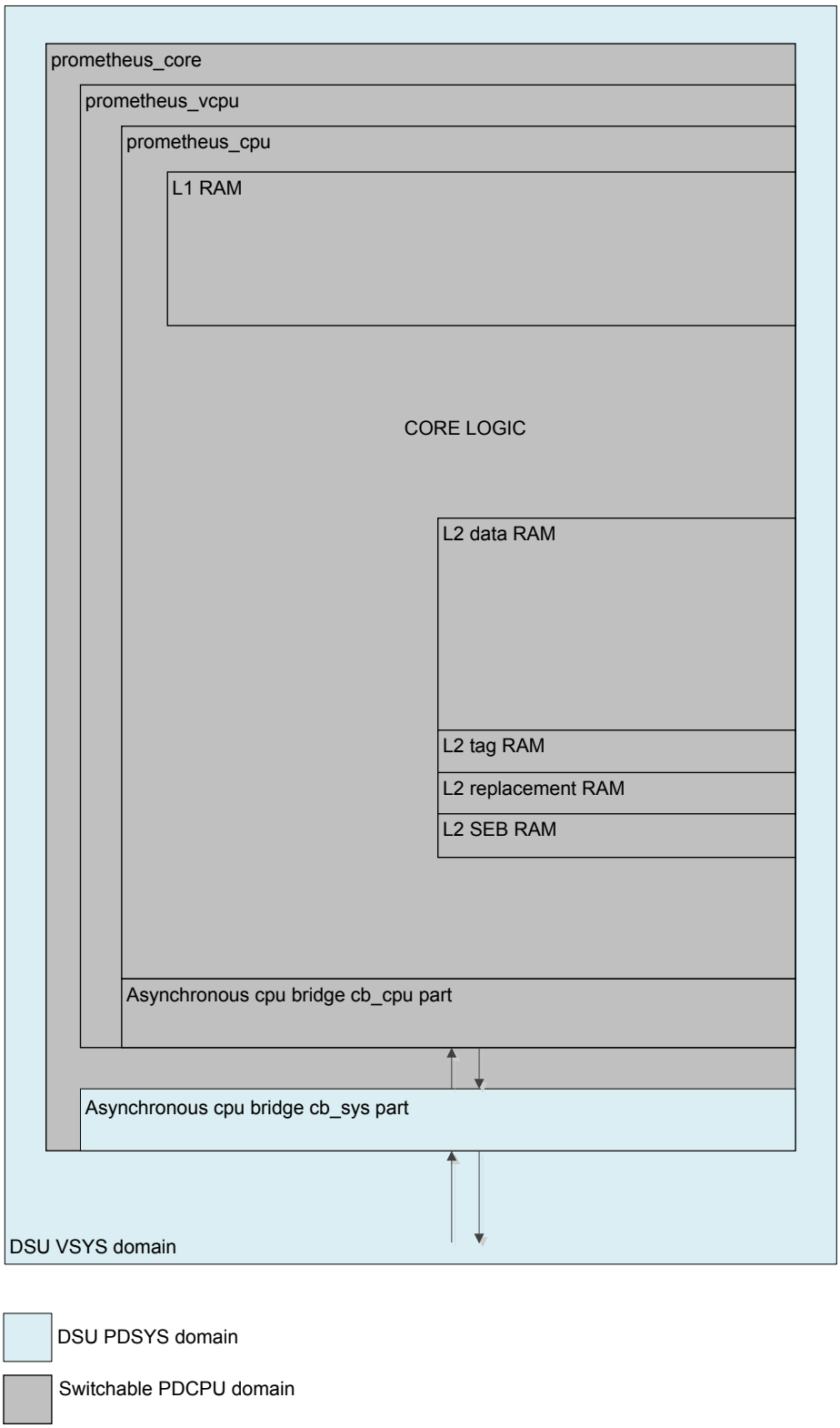

Figure A4-2 Cortex-A75 core power domain diagram at prometheus_core level

The following figure shows the power domains in the DSU, where everything in the same color is part of the same power domain. The example shows four Cortex-A75 cores. The number of Cortex-A75 cores can vary and the number of domains increase based on the number of Cortex-A75 cores present.

This example only shows the power domains that are associated with the Cortex-A75 cores, other power domains are required for a DSU.

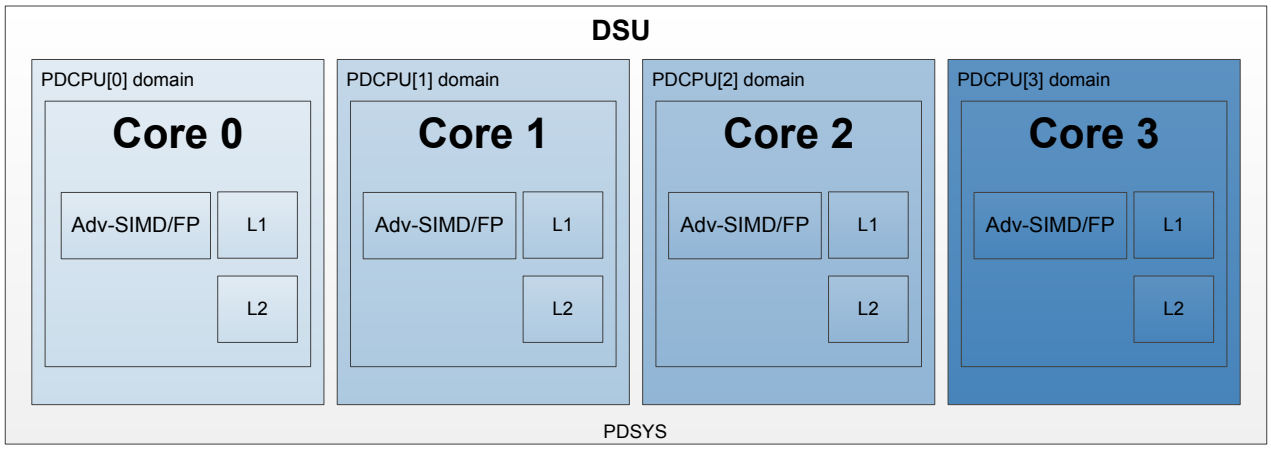

Figure A4-3 Cortex-A75 power domains at prometheus_core level

The following table shows the power domain that the Cortex-A75 core supports.

Table A4-1 Power domain description

Power domain	Description
PDCPU<n>	The domain includes the Advanced SIMD and floating-point block, the micro and main TLBs, L1 and L2 RAMs, and debug registers associated with the Cortex-A75 core. <n> is the number of Cortex-A75 cores. The number represents core 0, core 1, core 2, and core 3. If a core is not present, the corresponding power domain is not present.
PDSYS	Top level DSU power domain.

Clamping cells between power domains are inferred rather than instantiated in the RTL.

A4.4 Architectural clock gating modes

When the Cortex-A75 core is in standby mode, it is architecturally clock gated at the top of the clock tree.

Wait for Interrupt (WFI) and *Wait for Event* (WFE) are features of Armv8-A architecture that put the core in a low-power standby mode by architecturally disabling the clock at the top of the clock tree. The core is fully powered and retains all the state in standby mode.

A4.4.1 Core Wait for Interrupt

WFI puts the core in a low-power state by disabling most of the clocks in the core, while keeping the core powered up.

There is a small dynamic power overhead from the logic that is required to wake up the core from WFI low-power state. Other than this, the power that is drawn is reduced to static leakage current only.

When the core executes the `WFI` instruction, the core waits for all instructions in the core to retire before it enters low-power state. The `WFI` instruction ensures that all explicit memory accesses that occurred before the `WFI` instruction in program order have retired.

In addition, the `WFI` instruction ensures that store instructions have updated the cache or have been issued to the L3 memory system.

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

- An L3 snoop request that must be serviced by the core data caches.
- A cache or TLB maintenance operation that must be serviced by the core L1 instruction cache, data cache, TLB, or L2 cache.
- An APB access to the debug or trace registers residing in the core power domain.
- A GIC CPU access through the AXI4 stream channel.

Exit from WFI low-power state occurs when the core detects a reset or one of the WFI wake up events or when the AX14 stream channel accesses the GIC CPU interface. For more information, see the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

A4.4.2 Core Wait for Event

WFE is a feature of the Armv8-A architecture. It uses a locking mechanism based on events, to put the core in a low-power state by disabling most of the clocks in the core, while keeping the core powered up.

There is a small dynamic power overhead from the logic that is required to wake up the core from WFE low-power state. Other than this, the power that is drawn is reduced to static leakage current only.

A core enters into WFE low-power state by executing the `WFE` instruction. When the `WFE` instruction executes, the core waits for all instructions in the core to complete before it enters the idle or low-power state.

If the event register is set, execution of WFE does not cause entry into standby state, but clears the event register.

While the core is in WFE low-power state, the clocks in the core are temporarily enabled without causing the core to exit WFE low-power state when any of the following events are detected:

- An L3 snoop request that must be serviced by the core data caches.
- A cache or TLB maintenance operation that must be serviced by the core L1 instruction cache, data cache, TLB, or L2 cache.
- An APB access to the debug or trace registers residing in the core power domain.
- A GIC CPU access through the AXI4 stream channel.

Exit from WFE low-power state occurs on the assertion of the **EVENTI** input signal or one of the WFE wake-up events as described in the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

A4.5 Power control

All power mode transitions are performed at the request of the power controller, using a P-Channel interface to communicate with the Cortex-A75 core.

There is one P-Channel per core, plus one P-Channel for the cluster. The Cortex-A75 core provides the current requirements on the **PACTIVE** signals, so that the power controller can make decisions and request any change with **PREQ** and **PSTATE**. The Cortex-A75 core then performs any actions necessary to reach the requested power mode, such as gating clocks, flushing caches, or disabling coherency, before accepting the request.

If the request is not valid, either because of an incorrect transition or because the status has changed so that state is no longer appropriate, then the request is denied. The power mode of each core can be independent of other cores in the cluster, however the cluster power mode is linked to the mode of the cores.

A4.6 Core power modes

The following figure shows the supported modes for each core domain P-Channel, and the legal transitions between them.

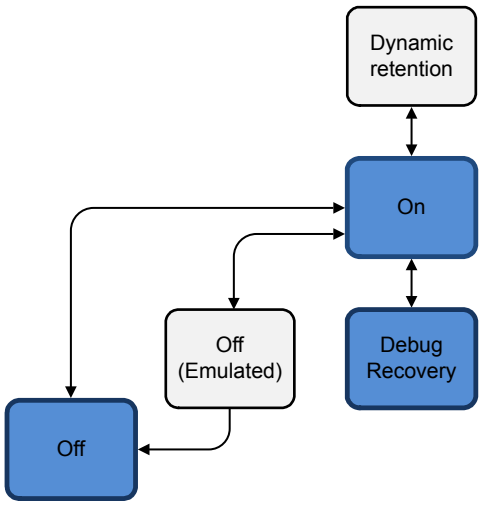

Figure A4-4 Cortex-A75 core power domain mode transitions

The blue modes indicate the modes the channel can be initialized into.

A4.6.1 On

In this mode, the core is on and fully operational.

The core can be initialized into the On mode. If the core does not use P-Channel, you can tie the core in the On mode by tying **PREQ** LOW.

When a transition to the On mode completes, all caches are accessible and coherent. Other than the normal architectural steps to enable caches, no additional software configuration is required.

When the core domain P-Channel is initialized into the On mode, either as a shortcut for entering that mode or as a tie-off for an unused P-Channel, it is an assumed transition from the Off mode. This includes an invalidation of any cache RAM within the core domain.

A4.6.2 Off

The Cortex-A75 core supports a full shutdown mode where power can be removed completely and no state is retained.

The shutdown can be for either the whole cluster or just for an individual core, which allows other cores in the cluster to continue operating.

In this mode, all core logic and RAMs are off. The domain is inoperable and all core state is lost. The L1 and L2 caches are disabled, flushed and the core is removed from coherency automatically on transition to Off mode.

A power-on reset can reset the core in this mode.

The core P-Channel can be initialized into this mode.

An attempted debug access when the core domain is off returns an error response on the internal debug interface indicating the core is not available.

A4.6.3 Off (emulated)

In this mode, all core domain logic and RAMs are kept on. However, core warm reset can be asserted externally to emulate a power off scenario while keeping core debug state and allowing debug access.

All debug registers must retain their mode and be accessible from the external debug interface. All other functional interfaces behave as if the core were Off.

A4.6.4 Core dynamic retention

In this mode, all core logic and RAMs are in retention and the core domain is inoperable. The core can be entered into this power mode when it is in WFI or WFE mode.

The core dynamic retention can be enabled and disabled separately for WFI and WFE by software running on the core. Separate timeout values can be programmed for entry into this mode from WFI and WFE mode:

- Use the CPUPWRCTLR.WFI_RET_CTRL register bits to program timeout values for entry into core dynamic retention mode from WFI mode.
- Use the CPUPWRCTLR.WFE_RET_CTRL register bits to program timeout values for entry into core dynamic retention mode from WFE mode.

When in dynamic retention and the core is synchronous to the cluster, the clock to the core is automatically gated outside of the domain. However, if the core is running asynchronous to the cluster, the system integrator must gate the clock externally during core dynamic retention. For more information, see the *Arm® DynamIQ™ Shared Unit Configuration and Sign-off Guide*.

The outputs of the domain must be isolated to prevent buffers without power from propagating unknown values to any operational parts of the system.

When the core is in dynamic retention there is support for Snoop, GIC, and debug access, so the core appears as if it were in WFI or WFE mode. When such an incoming access occurs, it stalls and the On **PACTIVE** bit is set HIGH. The incoming access proceeds when the domain is returned to On using the P-Channel.

When the incoming access completes, and if the core has not exited WFI or WFE mode, then the On **PACTIVE** bit is set LOW after the programmed retention timeout. The power controller can then request to reenter the core dynamic retention mode.

A4.6.5 Debug Recovery Mode

The debug recovery mode can be used to assist debug of external watchdog-triggered reset events.

It allows contents of the core L1 and L2 caches that were present before the reset to be observable after the reset. The contents of the caches are retained and are not altered on the transition back to the On mode.

By default, the core invalidates its caches when transitioning from OFF to an ON mode. If the P-Channel is initialized to the debug recovery mode, and the core is cycled through power-on reset along with system resets, then the cache invalidation is disabled. The cache contents are preserved when the core is transitioned to the On mode.

Debug recovery mode also supports preserving RAS state, in addition to the cache contents. In this case, a transition to the debug recovery mode is made from any of the current states. Once in debug recovery mode, a cluster-wide warm reset must be applied externally. The RAS and cache state are preserved when the core is transitioned to the On mode.

This mode is strictly for debug purposes. It must not be used for functional purposes, as correct operation of the caches is not guaranteed when entering this mode.

——— **Note** ———

- This mode can occur at any time with no guarantee of the state of the core. A P-Channel request of this type is accepted immediately, therefore its effects on the core, cluster, or the wider system are unpredictable, and a wider system reset might be required. In particular, if there were outstanding

memory system transactions at the time of the reset, then these may complete after the reset when the core is not expecting them and cause a system deadlock.

- If the system sends a snoop to the cluster during this mode, then depending on the cluster state, the snoop may get a response and disturb the contents of the caches, or it may not get a response and cause a system deadlock.

A4.7 Encoding for power modes

The following table shows the encodings for the supported modes for each core domain P-Channel.

Table A4-2 Core power modes COREPSTATE encoding

Power mode	Short name	PACTIVE bit number	PSTATE value [a]	Power mode description
Debug Recovery	DEBUG_RECOV	-	`0b001010`	Logic is off (or in reset), RAM state is retained and not invalidated when transition to On mode.
On	ON	8	`0b001000`	All powerup.
Core Dynamic Retention	FULL_RET	5	`0b000101`	Logic and RAM State are inoperable but retained.
Off (Emulated)	OFF_EMU	1	`0b000001`	On with Warm reset asserted, debug state is retained and accessible.
Off	OFF	0 (implicit)[b]	`0b000000`	All powerdown.

[a] PSTATE[5:4] are don't care.
[b] It is tied off to 0 and should be inferred when all other PACTIVE bits are LOW. For more information, see the *Low Power Interface Specification Arm® Q-Channel and P-Channel Interfaces*.

A4.8 Power domain states for power modes

The power domains can be controlled independently to give different combinations when powered-up and powered-down.

However, only some powered-up and powered-down domain combinations are valid and supported. The following information shows the supported power domain states for the Cortex-A75 core.

The PDCPU power domain supports the power states described in the following table.

Table A4-3 Power state description

Power state	Description
Off	Core off. Power to the block is gated.
Ret	Core retention. Logic and RAM retention power only.
On	Core on. Block is active.

Caution

States that are not shown in the following tables are unsupported and must not occur.

The following table describes the power modes, and the corresponding power domain states for individual cores. The power mode of each core is independent of all other cores in the cluster.

Table A4-4 Supported core power domain states

Power mode	Power domain state	Description
Debug recovery	On	Core on.
On	On	Core on. Advanced SIMD and floating-point block on.
Core dynamic retention	Ret	Core retention. Core logic and Advanced SIMD and floating-point block in retention.
Off (emulated)	On	Core on. Advanced SIMD and floating-point block on.
Off	Off	Core off.

Deviating from the legal power modes can lead to UNPREDICTABLE results. You must comply with the dynamic power management and powerup and powerdown sequences described in the following sections.

A4.9 Power up and down sequences

The following approach allows taking the Cortex-A75 cores in the cluster in and out of coherence.

Core powerdown

To take a core out of coherence ready for core powerdown, the following power down steps must be performed:

1. Save all architectural states.
2. Configure the GIC distributor to disable or reroute interrupts away from this core.
3. Set the CPUPWRCTLR.CORE_PWRDN_EN bit to 1 to indicate to the power controller that a powerdown is requested.
4. Execute an `ISB` instruction.
5. Execute a `WFI` instruction.

All L1 and L2 cache disabling, L1 and L2 cache flushing, and communication with the L3 memory system is performed in hardware after the `WFI` is executed, under the direction of the power controller.

Note

Executing any `WFI` instruction when the CPUPWRCTLR.CORE_PWRDN_EN bit is set automatically masks out all interrupts and wake-up events in the core. If executed when the CPUPWRCTLR.CORE_PWRDN_EN bit is set the WFI never wakes up and the core needs to be reset to restart.

For information about cluster powerdown, see the *Arm® DynamIQ™ Shared Unit Technical Reference Manual*.

Core powerup

To bring a core into coherence after reset, no software steps are required.

A4.10 Debug over powerdown

The Cortex-A75 core supports debug over powerdown, which allows a debugger to retain its connection with the core even when powered down. This enables debug to continue through powerdown scenarios, rather than having to re-establish a connection each time the core is powered up.

The debug over powerdown logic is part of the DebugBlock, which is external to the cluster, and must remain powered on during the debug over powerdown process.

See the *Arm® DynamIQ™ Shared Unit Technical Reference Manual*.

Chapter A5
Memory Management Unit

This chapter describes the *Memory Management Unit* (MMU) of the Cortex-A75 core.

It contains the following sections:

A5.1 About the MMU

The *Memory Management Unit* (MMU) is responsible for translating addresses of code and data *Virtual Addresses* (VA) to *Physical Addresses* (PAs) in the real system. In addition, the MMU controls actions such as memory access permissions, memory ordering, and cache policies for each region of memory.

A5.1.1 Main functions

The three main functions of the MMU are to:

- Control the table walk hardware that accesses translation tables in main memory.
- Translate *Virtual Addresses* (VAs) to *Physical Addresses* (PAs).
- Provide fine-grained memory system control through a set of virtual-to-physical address mappings and memory attributes that are held in translation tables.

Each stage of address translation uses a set of address translations and associated memory properties that are held in memory mapped tables called translation tables. Translation table entries can be cached into a *Translation Lookaside Buffer* (TLB).

The following table describes the components included in the MMU.

Table A5-1 TLBs and TLB caches in the MMU

Component	Description
Instruction micro TLB	32 entries, fully associative.
Data micro TLB	48 entries, fully associative.
Main TLB cache	1024 entries, 4-way set associative.
Secondary TLB cache	256 entries, 2-way set associative.
Translation table prefetcher	Detects access to contiguous translation tables and prefetches the next one. This prefetcher can be disabled in the ECTLR register.

The TLB entries contain either one or both of a global indicator and an *Address Space Identifier* (ASID) to permit context switches without requiring the TLB to be invalidated.

The TLB entries contain a *Virtual Machine Identifier* (VMID) to permit virtual machine switches by the hypervisor without requiring the TLB to be invalidated.

A5.1.2 AArch32 and AArch64 behavior differences

The Cortex-A75 core is an Armv8 compliant core that supports execution in both AArch32 and AArch64 states.

The following table shows the behavior differences between both execution states.

Table A5-2 AArch32 and AArch64 behavior differences

	AArch32	AArch64
Address translation system	The Armv8 address translation system resembles the Armv7 address translation system with *Large Physical Address Extension* (LPAE) and Virtualization Extensions.	The Armv8 address translation system resembles an extension to the Long descriptor format address translation system to support the expanded virtual and physical address space.
Translation granule	4KB for both *Virtual Memory System Architecture* (VMSA) and LPAE.	4KB, 16KB, or 64KB for LPAE.

Table A5-2 AArch32 and AArch64 behavior differences (continued)

	AArch32	AArch64
ASID size	8 bits.	8 or 16 bits depending on the value of TCR_ELx.AS.
VMID size	8 bits.	8 or 16 bits depending on the value of VTCR_EL2.VS.
PA size	44 bits only.	Maximum 44 bits. Any configuration of TCR_ELx.IPS over 44 bits is considered as 44 bits. You can enable or disable each stage of the address translation independently.

The Cortex-A75 core also supports the *Virtualization Host Extension* (VHE) including ASID space for EL2. When VHE is implemented and enabled, EL2 has the same behavior as EL1.

See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile* for more information on concatenated translation tables and for address translation formats.

A5.2 TLB organization

The TLB is a cache of recently executed page translations within the MMU. The Cortex-A75 core implements a two-level TLB structure. The TLB stores all page sizes and is responsible for breaking these down in to smaller pages when required for the data or instruction micro TLB.

A5.2.1 Instruction micro TLB

The instruction micro TLB is implemented as a 32-entry fully associative structure. This TLB caches entries at the 4KB, 16KB, 64KB, and 1MB granularity of VA to PA mapping only.

A hit in the instruction micro TLB provides a single **CLK** cycle access to the translation, and returns the PA to the instruction cache for comparison. It also checks the access permissions to signal a Prefetch Abort.

A5.2.2 Data micro TLB

The data micro TLB is a 48-entry fully associative TLB that is used by load and store operations. The cache entries have 4KB, 16KB, 64KB, and 1MB granularity of VA to PA mappings only.

A hit in the data micro TLB provides a single **CLK** cycle access to the translation, and returns the PA to the data cache for comparison. It also checks the access permissions to signal a Data Abort.

A5.2.3 Main TLB

The main TLB structure, which is shared by instruction and data, contains two cache RAMs. It handles misses from the instruction and data micro TLBs.

The following table describes the characteristics that apply to the main TLB.

Table A5-3 Characteristics of the main TLB

Characteristic	Note
4-way, set associative, 1024-entry cache	Stores VA to PA mappings for 4KB, 16KB, and 64KB page sizes.
2-way, set associative, 256-entry cache	Stores: • VA to PA mappings for 1MB, 2MB, 16MB, 32MB, 512MB, and 1GB block sizes. • *Intermediate physical address* (IPA) to PA mappings for 2MB (in a 4KB translation granule and in AArch32), 16MB (in a 16K translation granule), 512MB (in a 64K granule), and 2MB and 1GB (in 4K translation granule) block sizes. Only Non-secure EL1 and EL0 stage 2 translations are cached. • Intermediate PAs obtained during a translation table walk.
Support for both the VMSA and the LPAE.	-

Access to the main TLB usually takes four cycles. If a different page or block size mapping is used, then access to the main TLB can take longer.

The main TLB supports two translation table walks in parallel (two TLB misses), and can service a TLB lookup while the translation table walks are in progress. If there are three successive misses, the main TLB will stall.

Note

These two cache RAMs are invalidated automatically at reset unless the DISCACHEINVLD signal is set HIGH when the Cortex-A75 core is reset. This signal must only be used in diagnostic mode. If caches

are not invalidated on reset, their functionality cannot be guaranteed. See the *Arm® DynamIQ™ Shared Unit Technical Reference Manual* for more information on DISCACHEINVLD.

A5.3 TLB match process

The Armv8-A architecture provides support for multiple maps from the VA space that are translated differently.

TLB entries store the context information required to facilitate a match and avoid the need for a TLB flush on a context or virtual machine switch.

Each TLB entry contains a:

- VA.
- PA.
- Set of memory properties that include type and access permissions.

Each entry is either associated with a particular ASID or global. In addition, each TLB entry contains a field to store the VMID in the entry applicable to accesses from Non-secure EL0 and EL1 Exception levels.

Each entry is associated with a particular translation regime.

- EL3 in Secure state in AArch64 state only.
- EL2 or EL0 in Non-secure state.
- EL1 or EL0 in Secure state.
- EL1 or EL0 in Non-secure state.

A TLB match entry occurs when the following conditions are met:

- Its VA, moderated by the page size such as the VA bits[48:N], where N is $\log_2$ of the block size for that translation that is stored in the TLB entry, matches the requested address.
- Entry translation regime matches the current translation regime.
- The ASID matches the current ASID held in the CONTEXTIDR, TTBR0, or TTBR1 register, or the entry is marked global.
- The VMID matches the current VMID held in the VTTBR_EL2 register.
- The ASID and VMID matches are ignored when ASID and VMID are not relevant.
  ASID is relevant when the translation regime is:
  — EL2 in Non-secure state with HCR_EL2.E2H and HCR_EL2.TGE set to 1.
  — EL1 in Secure state.
  — EL1 in Non-secure state.

  VMID is relevant for EL1 in Non-secure state.

A5.4 Translation table walks

When the Cortex-A75 core generates a memory access, the MMU:

1. Performs a lookup in the micro TLB for the requested VA, current ASID, current VMID, and current translation regime in the relevant instruction or data.
2. If there is a miss in the relevant micro TLB, the MMU performs a lookup in the main TLB for the requested VA, current ASID, current VMID, and translation regime.
3. If there is a miss in the main TLB, the MMU performs a hardware translation table walk.

In the case of a main TLB miss, the hardware does a translation table walk as long as the MMU is enabled, and the translation using the base register has not been disabled.

If the translation table walk is disabled for a particular base register, the core returns a Translation Fault. If the TLB finds a matching entry, it uses the information in the entry as follows.

The access permission bits and the domain determine if the access is permitted. If the matching entry does not pass the permission checks, the MMU signals a Permission fault. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile* for details of Permission faults, including:

- A description of the various faults.
- The fault codes.
- Information regarding the registers where the fault codes are set.

Note

In AArch32 VMSA Short-descriptor format, the permission check includes the domain properties.

A5.4.1 AArch64 behavior

When executing in AArch64 state at a particular Exception level, you can configure the hardware translation table walk to use either the 4KB, 16KB, or 64KB translation granule. Program the Translation Granule bit, TG0, in the appropriate translation control register:

- TCR_EL1.
- TCR_EL2.
- TCR_EL3.
- VTCR_EL2.

For TCR_EL1, you can program the Translation Granule bits TG0 and TG1 to configure the translation granule respectively for TTBR0_EL1 and TTBR1_EL1, or TCR_EL2 when VHE is enabled.

A5.4.2 AArch32 behavior

When executing in AArch32 state in a particular mode, you can configure the MMU to perform hardware translation table walks using either the Short-descriptor translation table format, or the Long-descriptor translation table format. This is controlled by programming the *Extended Address Enable* (EAE) bit in the appropriate Secure or Non-secure *Translation Table Base Control Register* (TTBCR).

Note

Translations in Hyp mode are always performed with the Long-descriptor translation table format.

A5.5 MMU memory accesses

During a translation table walk, the MMU generates accesses. This section describes the specific behaviors of the core for MMU memory accesses.

A5.5.1 Configuring MMU accesses

By programming the IRGN and ORGN bits, you can configure the MMU to perform translation table walks in cacheable regions:

AArch32
- Translation table base registers (TTBR0/TTBR1_EL*x*) when using the Short-descriptor translation table format.
- TCR_EL*x* register when using the Long-descriptor translation table format.

AArch64 Appropriate TCR_EL*x* register.

If the encoding of both the ORGN and IRGN bits is Write-Back, the data cache lookup is performed and data is read from the data cache. External memory is accessed, if the ORGN and IRGN bit contain different attributes, or if the encoding of the ORGN and IRGN bits is Write-Through or Non-cacheable.

A5.5.2 Descriptor hardware update

The core supports hardware update in AArch64 state using hardware management of the access flag and hardware management of dirty state.

These features are enabled in registers TCR_ELx and VTCR_EL2.

Hardware management of the Access flag is enabled by the following configuration fields:

- TCR_ELx.HA for stage 1 translations.
- VTCR_EL2.HA for stage 2 translations.

Hardware management of dirty state is enabled by the following configuration fields:

- TCR_ELx.HD for stage 1 translations.
- VTCR_EL2.HD for stage 2 translations.

Note

Hardware management of dirty state can only be enabled if hardware management of the Access flag is enabled.

To support the hardware management of dirty state, the DBM field is added to the translation table descriptors as part of Armv8.1 architecture.

The core supports hardware update only in outer Write-Back and inner Write-Back memory regions.

If software requests a hardware update in a memory region that is not inner Write-Back or not outer Write-Back, then the core returns an abort with the following encoding:
- ESR.ELx.DFSC = `0b110001` for Data Aborts in AArch64.
- ESR.ELx.IFSC = `0b110001` for Instruction Aborts in AArch64.

A5.6 Specific behaviors on aborts and memory attributes

This section describes specific behaviors caused by aborts and also describes memory attributes.

MMU responses

When one of the following translation is completed, the MMU generates a response to the requester:

- A micro TLB hit.
- A main TLB hit.
- A translation table walk.

The response from the MMU contains the following information:

- The PA corresponding to the translation.
- A set of permissions.
- Domains information for AArch32 short descriptor format only.
- Secure or Non-secure.
- All the information required to report aborts. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile* for more details.

A5.6.1 External aborts

External aborts are defined as those that occur in the memory system rather than those that the MMU detects. Normally, external memory aborts are rare. External aborts are caused by errors flagged to the external interface.

When an external abort to the external interface occurs on an access for a translation table walk access, the MMU returns a synchronous external abort. For a load multiple or store multiple operation, the address captured in the fault address register is that of the address that generated the synchronous external abort.

See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile* for more information.

A5.6.2 Mis-programming contiguous hints

In the case of a mis-programming contiguous hint, when there is a descriptor that contains a set CH bit, all contiguous VAs contained in this block should be included in the input VA address space that is defined for stage 1 by TxSZ for TTBx or for stage 2 by {SL0, T0SZ}.

The Cortex-A75 core treats such a block (that might be a block within a contiguous set of blocks) as causing a Translation fault. This occurs even though the block is valid, and the address accessed within that block is within the size of the input address supported at a stage of translation.

A5.6.3 Conflict aborts

Conflict aborts are generated from the data or instruction micro TLB. If a conflict abort is detected in the main TLB, conflicting entries are invalidated and a translation table walk is executed.

See also the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile* for more information.

A5.6.4 Memory attributes

The memory region attributes specified in the TLB entry, or in the descriptor in case of translation table walk, determine if the access is:

- Normal Memory or Device type, Strongly ordered, or Device type when using the Short-descriptor format in AArch32 state.
- One of the three different device memory types that are defined for Armv8:

  Device-nGnRnE Device non-Gathering, non-Reordering, No Early Write Acknowledgment.

Device-nGnRE Device non-Gathering, non-Reordering, Early Write Acknowledgment.

Device-nGRE Device non-Gathering, Reordering, Early Write Acknowledgment.

In the Cortex-A75 core, a page is cacheable only if the inner memory attribute and outer memory attribute are Write Back. In all other cases, all pages are downgraded to Non-cacheable Normal memory.

When the MMU is disabled at stage 1 and stage 2, and SCTLR.I is set to 1, instruction prefetches are cached in the instruction cache but not in the unified cache. In all other cases, normal behavior on memory attribute applies.

See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile* for more information on translation table formats.

Chapter A6
Level 1 memory system

This chapter describes the L1 instruction cache and data cache that make up the L1 memory system.

It contains the following sections:

A6.1 About the L1 memory system

The Cortex-A75 L1 memory system is designed to enhance core performance and save power.

The L1 memory system consists of separate instruction and data caches. Both have a fixed size of 64KB.

A6.1.1 L1 instruction-side memory system

The L1 instruction memory system has the following key features:

- *Virtually Indexed, Physically Tagged* (VIPT), four-way set-associative instruction cache.
- Fixed cache line length of 64 bytes.
- Pseudo-random cache replacement policy.
- 128-bit read interface from the L2 memory system.

Note

In the L1 instruction memory system, aliases are not handled in hardware.

A6.1.2 L1 data-side memory system

The L1 data memory system has the following features:

- *Physically Indexed, Physically Tagged* (PIPT), 16-way set-associative L1 data cache.
- Fixed cache line length of 64 bytes.
- Pseudo-random cache replacement policy.
- 256-bit write interface to the L2 memory system.
- 128-bit read interface from the L2 memory system.
- Two 64-bit read paths from the data L1 memory system to the datapath.
- 128-bit write path from the datapath to the L1 memory system.

A6.2 Cache behavior

The implementation-specific features of the instruction and data caches include:

- At reset the instruction and data caches are disabled and both caches are automatically invalidated.

Note

The L1 instruction and data caches are invalidated automatically at reset unless the DISCACHEINVLD signal is set HIGH when the Cortex-A75 core is reset. This signal must only be used in diagnostic mode. If caches are not invalidated on reset, their functionality cannot be guaranteed. See the *Arm® DynamIQ™ Shared Unit Technical Reference Manual* for more information on DISCACHEINVLD.

- You can enable or disable each cache independently.
- Cache lockdown is not supported.
- On a cache miss, data for the cache linefill is requested in critical word-first order.

A6.2.1 Instruction cache disabled behavior

If the instruction cache is disabled, fetches cannot access any of the instruction cache arrays. An exception is the instruction cache operations. If the instruction cache is disabled, the instruction cache maintenance operations can still execute normally.

If the instruction cache is disabled, all instruction fetches to cacheable memory are treated as if they were non-cacheable. This treatment means that instruction fetches might not be coherent with caches in other cores, and software must take account of this.

A6.2.2 Instruction cache speculative memory accesses

Instruction fetches are speculative, as there can be several unresolved branches in the pipeline. There is no execution guarantee.

A branch instruction or exception in the code stream can cause a pipeline flush, discarding the currently fetched instructions. On instruction fetch accesses, pages with Device memory type attributes are treated as Non-Cacheable Normal Memory.

Device memory pages must be marked with the translation table descriptor attribute bit *Execute Never* (XN). The device and code address spaces must be separated in the physical memory map. This separation prevents speculative fetches to read-sensitive devices when address translation is disabled.

If the instruction cache is enabled, and if the instruction fetches miss in the L1 instruction cache, they can still look up in the L1 data caches. However, a new line is not allocated in the data cache unless the data cache is enabled.

See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile* for more information.

A6.2.3 Data cache disabled behavior

If the data cache is disabled, load and store instructions do not access any of the L1 data, L2 cache, and, if present, the DSU L3 cache arrays.

Unless the data cache is enabled, a new line is not allocated in the L2 or L3 caches due to an instruction fetch

Data cache maintenance operations are an exception. If the data cache is enabled, the data cache maintenance operations execute normally.

If the data cache is disabled, all loads and store instructions to cacheable memory are treated as if they were non-cacheable. Therefore, they are not coherent with the caches in this core or the caches in other cores, and software must take this into account.

The L2 and L1 data caches cannot be disabled independently.

A6.2.4 Data cache maintenance considerations

DCIMVAC operations in AArch32 state are treated as DCCIMVAC. DC IVAC operations in AArch64 state are treated as DC CIVAC except for permission checking and watchpoint matching.

In Non-secure state, DCISW operations in AArch32 state and DC ISW operations in AArch64 state, perform both a clean and invalidate of the target set/way. The values of HCR.SWIO and HCR_EL2.SWIO have no effect.

See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile* for more information.

A6.2.5 Data cache coherency

To maintain data coherency between multiple cores, the Cortex-A75 core uses the MESI protocol.

A6.2.6 Write streaming mode

A cache line is allocated to the L1 on either a read miss or a write miss.

However, there are some situations where allocating on writes is not required. For example, when executing the C standard library `memset`() function to clear a large block of memory to a known value. Writes of large blocks of data 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 counter this, the L1 memory system includes logic to detect when the core has stores pending to a full cache line when it is waiting for a linefill to complete, or when it detects a `DCZVA` (full cache line write to zero). If this situation is detected, then it switches into write streaming mode.

When in write streaming mode, loads behave as normal, and can still cause linefills, and writes still lookup in the cache, but if they miss then they write out to L2 (or possibly L3) rather than starting a linefill.

The L1 memory system continues in write streaming mode until it can no longer create a full cacheline of store (for example because of a lack of resource in the L1 memory system) or has detected a high proportion of store hitting in the cache.

Note

The L2 memory system is monitoring transaction traffic through L2 and, depending on different thresholds, can set a stream to go out of L2, L3, and L4.

The following registers control the different thresholds:

AArch32 state
CPUECTLR configures the L2, L3, and L4 write streaming mode threshold. See *B1.18 CPUECTLR, CPU Extended Control Register* on page B1-155.

AArch64 state
CPUECTLR_EL1 configure the L2, L3, and L4 write streaming mode threshold. See *B2.26 CPUECTLR_EL1, CPU Extended Control Register, EL1* on page B2-321.

A6.3 L1 instruction memory system

The L1 instruction side memory system provides an instruction stream to the decoder.

To increase overall performance and to reduce power consumption, it uses:
- Dynamic branch prediction.
- Instruction caching.

A6.3.1 Program flow prediction

The Cortex-A75 core contains program flow prediction hardware, also known as branch prediction.

Branch prediction increases overall performance and reduces power consumption. With program flow prediction disabled, all taken branches incur a penalty that is associated with flushing the pipeline.

To avoid this penalty, the branch prediction hardware predicts if a conditional or unconditional branch is to be taken. For conditional branches, the hardware predicts if the branch is to be taken. It also predicts the address that the branch goes to, known as the branch target address. For unconditional branches, only the target is predicted.

The hardware contains the following functionality:
- A BTAC holding the branch target address of previously taken branches.
- Dynamic branch predictor history.
- The return stack, a stack of nested subroutine return addresses.
- A static branch predictor.
- An indirect branch predictor.

Predicted and non-predicted instructions

Unless otherwise specified, the following list applies to A64, A32, and T32 instructions. As a rule the flow prediction hardware predicts all branch instructions regardless of the addressing mode, and includes:

- Conditional branches.
- Unconditional branches.
- Indirect branches that are associated with procedure call and return instructions.
- Branches that switch between A32 and T32 states.

The following branch instructions are not predicted:
- Data-processing instructions using the PC as a destination register.
- The `BXJ` instruction.
- Exception return instructions.

T32 state conditional branches

A T32 unconditional branch instruction can be made conditional by inclusion in an *If-Then* (IT) block. It is then treated as a conditional branch.

Return stack

The return stack stores the address and instruction set state.

This address is equal to the link register value stored in R14 in AArch32 state or X30 in AArch64 state.

The following instructions cause a return stack push if predicted:

- `BL r14`.
- `BLX (immediate)` in AArch32 state.
- `BLX (register)` in AArch32 state.
- `BLR` in AArch64 state.
- `MOV pc,r14`

In AArch32 state, the following instructions cause a return stack pop if predicted:

- `BX`
- `LDR pc, [r13], #imm`
- `LDM r13, {…pc}`
- `LDM r13, {…pc}`

In AArch64 state, the `RET` instruction causes a return stack pop.

As exception return instructions can change core privilege mode and security state, they are not predicted. These include:

- `LDM (exception return)`
- `RFE`
- `SUBS pc, lr`
- `ERET`

A6.4 L1 data memory system

The L1 data cache is organized as a *Physically Indexed, Physically tagged* (PIPT) cache featuring 16 ways.

Data cache invalidate on reset

The Armv8-A architecture does not support an operation to invalidate the entire data cache. If software requires this function, it must be constructed by iterating over the cache geometry and executing a series of individual invalidate by set/way instructions.

A6.4.1 Memory system implementation

This section describes the implementation of the L1 memory system.

Limited Order Regions

The Cortex-A75 core offers support for four limited ordering region descriptors, as introduced by the Armv8.1 Limited Ordering Regions. The core does not implement any optimization based on these descriptors.

Atomic instructions

The Cortex-A75 core supports the atomic instructions added in the Armv8.1 architecture.

Atomic instructions to cacheable memory can be performed as either near atomics or far atomics, depending on where the cache line containing the data resides. If the instruction hits in the L1 data cache in a unique state then it will be performed as a near atomic in the L1 memory system. If the atomic operation misses in the L1 cache, or the line is shared with another core then the atomic is sent as a far atomic out to the L3 cache. If the operation misses everywhere within the cluster, and the master interface is configured as CHI, and the interconnect supports far atomics, then the atomic will be passed on to the interconnect to perform the operation. If the operation hits anywhere inside the cluster, or the interconnect does not support atomics, then the L3 memory system will perform the atomic operation and allocate the line into the L3 cache if it is not already there.

Therefore, if software wants to ensure the atomic is performed as a near atomic, precede the atomic instruction with a `PLDW` or `PRFM PSTL1KEEP` instruction.

The Cortex-A75 core supports atomics to device or non-cacheable memory, however this relies on the interconnect also supporting atomics. If such an atomic instruction is executed when the interconnect does not support them, it will result in an asynchronous Data Abort.

LDAPR instructions

The core supports Load acquire instructions adhering to the RCpc consistency semantic introduced in the Armv8.3 extensions for A profile. This is reflected in register ID_AA64ISAR1_EL1 where bits[23:20] are set to `0b0001` to indicate that the core supports `LDAPRB`, `LDAPRH`, and `LDAPR` instructions implemented in AArch64.

Transient memory region

The core has a specific behavior for memory regions that are marked as Write-Back cacheable and transient, as defined in the Armv8 architecture.

For any load that is targeted at a memory region that is marked as transient, the following occurs:

- If the memory access misses in the L1 data cache, the returned cache line is allocated in the L1 data cache but is marked as transient.
- On eviction, if the line is clean and marked as transient, it is not allocated into the L2 cache but is marked as invalid.

For streams of contiguous stores that are targeted at a memory region that is marked as transient, the following occurs:

- If a full cache line is written and misses in the L1 data cache, the complete cache line is streamed to the external memory system without being allocated into the L1 data cache, the L2 cache or, if present, in the L3 cache.

Non-temporal hints

Memory accesses (load/store pair and prefetch hint) with non-temporal hints behave as normal memory accesses that are targeted at a memory region marked as transient.

A6.4.2 Internal exclusive monitor

The Cortex-A75 core L1 memory system has an internal exclusive monitor.

This monitor is a 2-state, open and exclusive, state machine that manages Load-Exclusive or Store-Exclusive accesses and Clear-Exclusive (`CLREX`) instructions. You can use these instructions to construct semaphores, ensuring synchronization between different processes running on the core, and also between different cores that are using the same coherent memory locations for the semaphore. A Load-Exclusive instruction tags a small block of memory for exclusive access. CTR.ERG defines the size of the tagged block as 16 words, one cache line.

——— **Note** ———

A load/store exclusive instruction is any one of the following:

- In the A64 instruction set, any instruction that has a mnemonic starting with `LDX`, `LDAX`, `STX`, or `STLX`.
- In the A32 and T32 instruction sets, any instruction that has a mnemonic starting with `LDREX`, `STREX`, `LDAEX`, or `STLEX`.

———————————

See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile* for more information about these instructions.

A6.5 Data prefetching

This section describes the data prefetching behavior for the Cortex-A75 core.

Preload instructions

The Cortex-A75 core supports the AArch64 *Prefetch Memory* (`PRFM`) instructions and the AArch32 *Prefetch Data* (`PLD`) and *Preload Data With Intent To Write* (`PLDW`) instructions. These instructions signal to the memory system that memory accesses from a specified address are likely to occur soon. The memory system acts by taking actions that aim to reduce the latency of the memory access when they occur. `PRFM` instructions perform a lookup in the cache, and if they miss and are to a cacheable address, a linefill starts. However, the `PRFM` instruction retires when its linefill is started, rather than waiting for the linefill to complete. This enables other instructions to execute while the linefill continues in the background.

The *Preload Instruction* (`PLI`) memory system hint performs preloading in the L2 cache for cacheable accesses if they miss in both the L1 instruction cache and L2 cache. Instruction preloading is performed in the background.

For more information about prefetch memory and preloading caches, see the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

Data prefetching and monitoring

The data cache implements an automatic prefetcher that monitors cache misses in the core.

When a pattern is detected, the automatic prefetcher starts linefills in the background. The prefetcher recognizes a sequence of data cache misses at a fixed stride pattern that lies in 32 cache lines, plus or minus. Any intervening stores or loads that hit in the data cache do not interfere with the recognition of the cache miss pattern. Up to twelve independent streams can be handled by the prefetcher.

This L1 prefetcher prefetches into either the L1 or L2, or in the DSU L3 cache if present.

A spatial prefetcher prefetches into the L2 cache.

The CPUECTLR register allows you to have some control over the prefetcher. See *B1.18 CPUECTLR, CPU Extended Control Register* on page B1-155 for more information on the control of the prefetcher.

Use the prefetch memory system instructions for data prefetching where short sequences or irregular pattern fetches are required.

Data cache zero

The Armv8-A architecture introduces a *Data Cache Zero by Virtual Address* (`DC ZVA`) instruction.

In the Cortex-A75 core, this enables a block of 64 bytes in memory, aligned to 64 bytes in size, to be set to zero.

For more information, see the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

A6.6 Direct access to internal memory

The Cortex-A75 core provides a mechanism to read the internal memory that is used by the L1 cache and TLB structures through implementation defined system registers. This functionality can be useful when investigating issues where the coherency between the data in the cache and data in system memory is broken.

When the core executes in AArch64 state, the appropriate memory block and location are selected using several write-only registers. The data is read from read-only registers as shown in the following table. These operations are available only in EL3. In all other modes, executing these instructions results in an Undefined Instruction exception.

Table A6-1 AArch64 registers used to access internal memory

Register name	Function	Access	Operation	Rd Data
CDBGDR0_EL3	Data Register 0	Read-only	`MRS <Xd>, S3_6_c15_c0_0`	Data
CDBGDR1_EL3	Data Register 1	Read-only	`MRS <Xd>, S3_6_c15_c0_1`	Data
CDBGDR2_EL3	Data Register 2	Read-only	`MRS <Xd>, S3_6_c15_c0_2`	Data
CDBGDCT_EL3	Data Cache Tag Read Operation Register	Write-only	`MSR S1_6_c15_c2_0, <Xd>`	Set/Way
CDBGICT_EL3	Instruction Cache Tag Read Operation Register	Write-only	`MSR S1_6_c15_c2_1, <Xd>`	Set/Way
CDBGTT_EL3	TLB Tag Read Operation Register	Write-only	`MSR S1_6_c15_c2_2, <Xd>`	Index/Way
CDBGDCD_EL3	Data Cache Data Read Operation Register	Write-only	`MSR S1_6_c15_c4_0, <Xd>`	Set/Way/Offset
CDBGICD_EL3	Instruction Cache Data Read Operation Register	Write-only	`MSR S1_6_c15_c4_1, <Xd>`	Set/Way/Offset
CDBGTD_EL3	TLB Data Read Operation Register	Write-only	`MSR S1_6_c15_c4_2, <Xd>`	Index/Way

When the core executes in AArch32 state, the appropriate memory block and location are selected using several write-only system registers. The data is read from read-only system registers as shown in the following table. These operations are available only in EL3. In all other modes, executing the system operation results in an Undefined Instruction exception.

Table A6-2 AArch32 CP15 registers used to access internal memory

Register name	Function	Access	CP15 operation	Rd Data
CDBGDR0	Data Register 0	Read-only	`MRC p15, 6, <Rd>, c15, c0, 0`	Data
CDBGDR1	Data Register 1	Read-only	`MRC p15, 6, <Rd>, c15, c0, 1`	Data
CDBGDR2	Data Register 2	Read-only	`MRC p15, 6, <Rd>, c15, c0, 2`	Data
CDBGDCT	Data Cache Tag Read Operation Register	Write-only	`MCR p15, 6, <Rd>, c15, c2, 0`	Set/Way
CDBGICT	Instruction Cache Tag Read Operation Register	Write-only	`MCR p15, 6, <Rd>, c15, c2, 1`	Set/Way
CDBGTT	TLB Tag Read Operation Register	Write-only	`MCR p15, 6, <Rd>, c15, c2, 2`	Index/Way
CDBGDCD	Data Cache Data Read Operation Register	Write-only	`MCR p15, 6, <Rd>, c15, c4, 0`	Set/Way/Offset
CDBGICD	Instruction Cache Data Read Operation Register	Write-only	`MCR p15, 6, <Rd>, c15, c4, 1`	Set/Way/Offset
CDBGTD	TLB Data Read Operation Register	Write-only	`MCR p15, 6, <Rd>, c15, c4, 2`	Index/Way

A6.6.1 Encoding for tag and data in the L1 data cache

The core data cache consists of a 16-way set-associative structure.

The encoding, set in Rd in the appropriate CP15 write instruction, used to locate the required cache data entry for tag and data memory is shown in the following table. It is similar for both the tag and data RAM access. Data RAM access includes an additional field to locate the appropriate doubleword in the cache line.

Tag RAM encoding includes an additional field to select which one of the two cache channels must be used to perform any access.

Table A6-3 Data cache tag location encoding

Bit fields of Rd	Description
[31:28]	Cache way
[27:13]	Unused
[12]	Cache channel
[11:6]	Cache index
[5:0]	Unused

Table A6-4 Data cache data location encoding

Bit fields of Rd	Description
[31:28]	Cache way
[27:12]	Unused
[11:6]	Set index
[5:3]	Cache doubleword data offset
[2:0]	Unused

Data cache reads return 64 bits of data in Data Register 0 and Data Register 1. If cache protection is supported, Data Register 2 is used to report ECC information using the format shown in the following table.

Table A6-5 Data cache tag format

Register	Bit field	Description
Data Register 0	[31:10]	TAG[33:12].
	[9:7]	RRIP.
	[6:4]	Memory attribute.
	[3:2]	Line state [UC].
	[1:0]	Unused.
Data Register 1	[31:11]	Unused.
	[10]	Address Non-secure bit.
	[9:0]	TAG[43:34].

Table A6-5 Data cache tag format (continued)

Register	Bit field	Description
Data Register 2	[31:7]	Unused.
	[6:0]	ECC that corresponds to the word {Register_1[10:0], Register_0[31:0]} for implementations that support cache protection. Unused for implementations that do not support cache protection.

Table A6-6 Data cache data format

Register	Bit field	Description
Data Register 0	[31:0]	Bits [n*64+31:n*64] of the selected line. n is the double word offset.
Data Register 1	[31:0]	Bits [n*64+63:n*64+32] of the selected line. n is the double word offset.
Data Register 2	[31:14]	Unused.
	[13:7]	ECC that correspond to the word on Register 1 for implementations that support cache protection. Unused for implementations that do not support cache protection.
	[6:0]	ECC that correspond to the word on Register 0 for implementations that support cache protection. Unused for implementations that do not support cache protection.

A6.6.2 Encoding for tag and data in the L1 instruction cache

The following tables show the encoding required to select a given cache line.

Table A6-7 Instruction cache tag location encoding

Bit fields of Rd	Description
[31:30]	Cache Way
[29:14]	Unused
[13:6]	Index
[5:0]	Unused

Table A6-8 Instruction cache data location encoding

Bit fields of Rd	Description
[31:30]	Cache Way
[29:14]	Unused

Table A6-8 Instruction cache data location encoding (continued)

Bit fields of Rd	Description
[13:6]	Index
[5:3]	Cache doubleword data offset, Data Register only
[2:0]	Unused

The CP15 Instruction Cache Tag Read Operation returns one cache tag entry and stores it in three registers, Data Registers 0-2.

The following table shows the tag bit format of Data Registers 0-2.

Table A6-9 Instruction cache tag format

Register	Bit fields	Description
Rd0	[31:0]	Tag address
Rd1	[31:2]	Unused
	[1]	Valid bit
	[0]	Non-secure (NS) state
Rd2	[31:2]	Unused
	[1:0]	Parity bits

The CP15 Instruction Cache Data Read Operation returns a 64-bit entry from the cache in Data Registers 0-2.

The following table shows the data bit format of Data Registers 0-2.

Table A6-10 Instruction cache data format

Register	Bit fields	Description
Rd0	[31:0]	Data [31:0]
Rd1	[31:0]	Data [63:32]
Rd2	[31:2]	Unused
	[1:0]	Parity bits

A6.6.3 Encoding for the TLB

The main TLB is built from two cache RAMs:

RAM0 This is a 4-way set-associative RAM array for 4KB, 16KB, and 64KB page translation entries.

RAM1 This is a 2-way set-associative RAM array for 1MB, 2MB, 16MB, 32MB, 512MB, and 1GB block sizes, walk cache, and IPA cache.
RAM1 holds:

- TLB regular format entries.
- TLB walk format entries.
- TLB IPA format entries.

The TLB walk format entry is identified by the Entry type field set to `1`.

The TLB IPA format entry is identified by the Entry type field set to `0`, the S1 translation mode field set to `0b01`, and the S1 level field set to `0b10`.

To read the individual entries into the data registers, software must write to the TLB Data Read Operation Register. The following table shows the write TLB Data Read Operation Register location encoding.

Table A6-11 TLB Data Read Operation Register location encoding

Bits	Description
[31:30][c]	TLB way
[29:9]	Unused
[8]	Type: `0` RAM0. `1` RAM1.
[7:0][d]	TLB index.

For direct RAM access to the TLB tag RAM, CDBGTT, tag RAM0 and RAM1 are 78-bit wide. The data is returned in the following registers:

Data Register 0[31:0] Tag RAM data[31:0]
Data Register 1[31:0] Tag RAM data[61:32]
Data Register 1[15:0] Tag RAM data[77:62]

For direct RAM access to the TLB data RAM, CDBGTD, data RAM0 and RAM1 are 54-bit wide. The data is returned in the following registers:

Data Register 0[31:0]
Data RAM data[31:0]
Data Register 1[21:0]
Data RAM data[53:32]

Memory attributes

TLB encoding descriptions are detailed in the following sections:

- *A6.6.4 TLB regular format* on page A6-85.
- *A6.6.5 Encoding for intermediate translation entries* on page A6-87.
- *A6.6.6 Encoding for IPA cache* on page A6-90.

The following table gives the memory attribute bit descriptions that apply to the memory attribute bits identified in the TLB encoding descriptions.

[c] Bit [31] is unused if bit [8] = `0b1`.
[d] Bits [7:6] are unused if bit [8] = `0b1`.

Table A6-12 Memory attributes

	Cacheable	Non cacheable	nGRE	nGnRE	nGnRnE
Mem_attr[3]	Inner transient hint	0	Device from stage 2	Device from stage 2	Device from stage 2
Mem_attr[2]	Shareable	1	1	0	0
Mem_attr[1]	Outer allocate hint	1	0	1	0
Mem_attr[0]	1	0	0	0	0

A6.6.4 TLB regular format

The following section describes the encoding for TLB direct RAM accesses.

The following table shows the TLB data RAM regular format encoding.

Table A6-13 TLB data RAM regular format encoding

Bits	Name	Description
[53]	TLB data parity	Indicates the parity protection for the TLB data RAM.
[52:51]	-	RES0.
[50]	Split hint	Indicates whether stage 2 page size is smaller than stage 1 page size.
[49]	Intermediate translation hint	Indicates whether the entry is an intermediate translation.
[48:45]	Memory attribute	See *Table A6-12 Memory attributes* on page A6-85.
[44]	Stage 2 DBM	Indicates the dirty bit memory from final stage 2 descriptor.
[43:42]	HAP[1:0]	Indicates the hypervisor access permissions from stage 2 translation.
[41:40]	S2 level	Indicates the stage 2 translation level. `00` Level 3. `01` Level 2. `10` Level 1. `00` Stage 2 is off.
[39]	S1 PXN	Indicates the Privilege eXecute Never attribute from stage 1.
[38]	S1 UXN	Indicates the User eXecute Never attribute from stage 1.
[37]	S2 PXN	Indicates the Privilege eXecute Never attribute from stage 2.
[36]	S2 UXN	Indicates the User eXecute Never attribute from stage 2.
[35:33]	AP[2:0]	Indicates the access permission from stage 1. VMSA : AP[2:0] is fully used with AFE disabled. VMSA: AP[0] is not used with AFE enabled. LPAE: AP[0] is DBM of stage 1 descriptor.
[32]	NS	Indicates the Non-secure VA.
[31:0]	PA[43:12]	Indicates the PA.

The following table shows the TLB tag RAM regular format encoding.

Table A6-14 TLB tag RAM regular format encoding

Bits	Name	Description
[77:76]	TLB tag parity	Indicates the parity protection for the TLB tag RAM.
[75]	Valid	Indicates whether the entry is valid.
[74:42]	VA[48:12]	Indicates the VA.
[41:26]	VMID	Indicates the virtual machine identifier.
[25:22]	ASID[15:12]	**LPAE** Address space identifier. **VMSA:** RES0.
[21:18]	ASID[11:8] Domain index	**LPAE** Address space identifier. **VMSA:** Domain index.
[17:10]	ASID[7:0]	Indicates the address space identifier.
[9]	NG	Indicates the Non-global bit.
[8:6]	Page size	Indicates the size of memory mapped by block or page that is allocated in the TLB entry. **TLB RAM0** `0b000` 4KB. `0b001` 16KB. `0b010` 64KB. **TLB RAM1** `0b000` 1MB. `0b001` 2MB. `0b010` 16MB. `0b011` 32MB. `0b100` 512MB. `0b101` 1GB.

[5:4]	Translation regime	The possible values are:
		`0b00`
		• In AArch64, EL0 and EL1 are Non-secure.
		• In AArch32, PL0 and PL1 are Non-secure.
		`0b01`
		• In AArch64, EL0 and EL1 are Secure.
		• In AArch32, PL0 and PL1 are Secure.
		`0b10`
		• In AArch64, EL2 is Non-secure.
		• In AArch32, PL2 is Non-secure.
		`0b11`
		EL3 is Secure.

Table A6-14 TLB tag RAM regular format encoding (continued)

Bits	Name	Description

[3:2]	S1 Translation Mode	Indicates the stage 1 translation mode.
		`0b00`
		VMSA short format.
		`0b01`
		LPAE 16KB.
		`0b10`
		LPAE 4KB.
		`0b11`
		LPAE 64KB.

[1:0]	S1 level	Indicates the stage 1 level that gave this translation for AArch32. **VMSA** `0b00` Level 2. `0b01` Level 1. `0b10` Level 1 with contiguous hint. `0b11` Level 2 with contiguous hint. **LPAE 4KB** `0b00` Level 3. `0b01` Level 2. `0b10` Level 1. `0b11` Level 3 with contiguous hint. **LPAE 16KB** `0b00` Level 2. `0b01` Level 1. `0b10` Unused in regular format. `0b11` Level 2 with contiguous hint. **LPAE 64KB** `0b00` Level 2. `0b01` Level 1. `0b10` Unused in regular format. `0b11` Level 2 with contiguous hint.

A6.6.5 Encoding for intermediate translation entries

The encoding for intermediate translation entries only applies to RAM1.

The following table shows the encoding for an intermediate translation entry in the TLB data RAM1.

Table A6-15 TLB data RAM1 intermediate translation entries encoding

Bits	Name	Description
[53]	TLB data parity	Indicates the parity protection for the TLB data RAM.
[52:51]	-	RES0.

Table A6-15 TLB data RAM1 intermediate translation entries encoding (continued)

Bits	Name	Description
[50]	Split hint	Indicates whether stage 2 page size is smaller than stage 1 page size. This bit is RES0 on walk entries.
[49]	Intermediate translation hint	Indicates whether the entry is an intermediate translation.
[48:45]	Memory attribute	*Table A6-12 Memory attributes* on page A6-85.
[44]	Stage 2 DBM	Not applicable.
[43:42]	PA[11:10]	Indicates the descriptor PA.
[41:40]	S2 level	Not applicable.
[39]	S1 PXNTable	Indicates the Privilege eXecute Never attribute from stage 1.
[38]	S1 UXNTable	Indicates the User eXecute Never attribute from stage 1.
[37]	S2 PXN	Not applicable.
[36]	S2 UXN	Not applicable.
[35]	-	RES0.
[34:33]	APTable[2:0]	Indicates the descriptor access permission bits. In configurations where these bits are not relevant, for example where hierarchical permission is disabled, the bits are clear.
[32]	NSTable	Indicates the Non-secure VA.
[31:0]	PA[43:12]	Indicates the descriptor PA.

The following table shows the encoding for an intermediate translation entry in the TLB tag RAM1.

Table A6-16 TLB tag RAM1 intermediate translation entries encoding

Bits	Name	Description
[77:76]	TLB tag parity	Indicates the parity protection for the TLB tag RAM.
[75]	Valid	Indicates whether the entry is valid.
[74:42]	VA[48:16]	Indicates the VA.
[41:26]	VMID	Indicates the virtual machine identifier.
[25:22]	ASID[15:12]	**LPAE** Address space identifier. **VMSA:** RES0.
[21:18]	ASID[11:8] Domain index	**LPAE** Address space identifier. **VMSA:** Domain index.

Table A6-16 TLB tag RAM1 intermediate translation entries encoding (continued)

Bits	Name	Description
[17:10]	ASID[7:0]	Indicates the address space identifier.
[9]	NG	Indicates the Non-global bit.
[8:6]	Page size	Indicates the size of memory mapped by block or page that is allocated in the TLB entry. **TLB RAM1** `0b000` 1MB. `0b001` 2MB. `0b010` 16MB. `0b011` 32MB. `0b100` 512MB. `0b101` 1GB.

[5:4]	Translation regime	The possible values are:
		`0b00`
		• In AArch64, EL0 and EL1 are Non-secure.
		• In AArch32, PL0 and PL1 are Non-secure.
		`0b01`
		• In AArch64, EL0 and EL1 are Secure.
		• In AArch32, PL0 and PL1 are Secure.
		`0b10`
		• In AArch64, EL2 is Non-secure.
		• In AArch32, PL2 is Non-secure.
		`0b11`
		EL3 is Secure.

Table A6-16 TLB tag RAM1 intermediate translation entries encoding (continued)

Bits	Name	Description
[3:2]	S1 Translation Mode	Indicates the stage 1 translation mode. `0b00` VMSA short format. `0b01` LPAE 16KB. `0b10` LPAE 4KB. `0b11` LPAE 64KB.
[1:0]	S1 level	Indicates the stage 1 level that gave this translation for AArch32. **VMSA** `0b00` Level 2. `0b01` Level 1. `0b10` Level 1 with contiguous hint. `0b11` Level 2 with contiguous hint. **LPAE 4KB** `0b00` Level 3. `0b01` Level 2. `0b10` Level 1. `0b11` Level 3 with contiguous hint. **LPAE 16KB** `0b00` Level 2. `0b01` Level 1. `0b10` Unused in regular format. `0b11` Level 2 with contiguous hint. **LPAE 64KB** `0b00` Level 2. `0b01` Level 1. `0b10` Unused in regular format. `0b11` Level 2 with contiguous hint.

A6.6.6 Encoding for IPA cache

The encoding for IPA cache only applies to RAM1.

The following table shows the TLB data RAM encoding for an IPA translation entry in RAM1.

Table A6-17 TLB data RAM1 IPA translation entries encoding

Bits	Name	Description
[53]	TLB data parity	Indicates the parity protection for the TLB data RAM.
[52:49]	-	RES0.
[48:45]	S2 Memory attribute	*Table A6-12 Memory attributes* on page A6-85.
[44]	Stage 2 DBM	Indicates the dirty bit memory from final stage 2 descriptor.
[43:42]	S2 HAP	Indicates the stage 2 permission bits.
[41:40]	S2 level	Indicates the stage 2 translation level. `00` Level 3. `01` Level 2. `10` Level 1. `00` Stage 2 is off.
[39]	S1 PXNTable	Not applicable.
[38]	S1 UXNTable	Not applicable.
[37]	S2 PXN	Indicates the Privilege eXecute Never attribute from stage 2.
[36]	S2 UXN	Indicates the User eXecute Never attribute from stage 2.
[35]	-	RES0.
[34:33]	APTable[1:0]	Not applicable.
[32]	NS	Indicates the Non-secure VA.
[31:0]	PA[43:12]	Indicates the descriptor PA.

The following table shows the TLB tag RAM encoding for an IPA translation entry in RAM1.

Table A6-18 TLB tag RAM1 IPA translation entries encoding

Bits	Name	Description
[77:76]	TLB tag parity	Indicates the parity protection for the TLB tag RAM.
[75]	Valid	Indicates whether the entry is valid.
[74:70]	-	RES0.
[69:42]	IPA[43:16]	Indicates the intermediate physical address.
[41:26]	VMID	Indicates the virtual machine identifier.
[25:22]	ASID[15:12]	Not applicable.
[21:18]	ASID[11:8] Domain index	Not applicable.
[17:10]	ASID[7:0]	Not applicable.

Table A6-18 TLB tag RAM1 IPA translation entries encoding (continued)

Bits	Name	Description
[9]	NG	Not applicable.
[8:6]	Page size	Indicates the size of memory mapped by block or page that is allocated in the TLB entry. **TLB RAM1** `0b001` 2MB in LPAE 4KB granule. `0b011` 32MB in LPAE 16KB granule. `0b100` 512MB in LPAE 64KB granule. `0b101` 1GB in LPAE 4KB granule.
[5:4]	Translation regime	The possible value is: `0b00` • In AArch64, EL0 and EL1 are Non-secure. • In AArch32, PL0 and PL1 are Non-secure.
[3:2]	S1 Translation Mode	Indicates the stage 1 translation mode. The value is always `0b01`.
[1:0]	S1 level	Indicates the stage 1 level that gave this translation for AArch32. The value is always `0b10`.

Chapter A7
Level 2 memory system

This chapter describes the L2 memory system.

It contains the following sections:

A7.1 About the L2 memory system

The L2 memory subsystem consist of:

- An 8-way set associative L2 cache with a configurable size of 256KB or 512KB. Cache lines have a fixed length of 64 bytes.
- Optional ECC protection for all RAM structures.
- A spatial prefetcher.
- Interfaces to both the L1 memory system and the DSU.

A7.2 About the L2 cache

The integrated L2 cache is the Point of Unification for the Cortex-A75 core. It handles both instruction and data requests from the instruction side and data side of each core respectively.

When fetched from the system, instructions are allocated to the L2 cache and can be invalidated during maintenance operations.

The L2 cache is invalidated automatically at reset unless the **DISCACHEINVLD** signal is set HIGH when the Cortex-A75 core is reset. This signal must be used only in diagnostic mode. If caches are not invalidated on reset, their functionality cannot be guaranteed. See the *Arm® DynamIQ™ Shared Unit Technical Reference Manual* for more information on the **DISCACHEINVLD** signal.

A7.3 Support for memory types

The Cortex-A75 core simplifies the coherency logic by downgrading some memory types.

- Memory that is marked as both Inner Write-Back Cacheable and Outer Write-Back Cacheable is cached in the L1 data cache and the L2 cache.
- Memory that is marked Inner Write-Through is downgraded to Non-cacheable.
- Memory that is marked Outer Write-Through or Outer Non-cacheable is downgraded to Non-cacheable, even if the inner attributes are Write-Back cacheable.

Chapter A8
Reliability, Availability, and Serviceability (RAS)

This chapter describes the RAS features implemented in the Cortex-A75 core.

It contains the following sections:

A8.1 Cache ECC and parity

The Cortex-A75 core implements the RAS extension to the Arm architecture which provides mechanisms for standardized reporting of the errors generated by cache protection mechanisms.

When configured with core cache protection, the Cortex-A75 core can detect and correct a 1-bit error in any RAM and detect 2-bit errors in some RAMs.

Note

For information about SCU-L3 cache protection, see the *Arm® DynamIQ™ Shared Unit Technical Reference Manual*.

The RAS extension improves the system by reducing unplanned outages:

- Transient errors can be detected and corrected before they cause application or system failure.
- Failing components can be identified and replaced.
- Failure can be predicted ahead of time to allow replacement during planned maintenance.

Errors that are present but not detected are known as latent or undetected errors. A transaction carrying a latent error is corrupted. In a system with no error detection, all errors are latent errors and are silently propagated by components until either:

- They are masked and do not affect the outcome of the system. These are benign or false errors.
- They affect the service interface of the system and cause failure. These are silent data corruptions.

The severity of a failure can range from minor to catastrophic. In many systems, data or service loss is regarded as more of a minor failure than data corruption, as long as backup data is available.

The RAS extension focuses on errors that are produced from hardware faults, which fall into two main categories:

- Transient faults.
- Persistent faults.

The RAS extension describes data corruption faults, which mostly occur in memories and on data links. RAS concepts can also be used for the management of other types of physical faults found in systems, such as lock-step errors, thermal trip, and mechanical failure. The RAS extension provides a common programmers model and mechanisms for fault handling and error recovery.

A8.2 Cache protection behavior

The configuration of the RAS extension that is implemented in the Cortex-A75 core includes cache protection.

In this case, the Cortex-A75 core protects against errors that result in a RAM bitcell holding the incorrect value.

The RAMs in the Cortex-A75 core have the following capability:

SED
Single Error Detect. One bit of parity is applicable to the entire word. The word size is specific for each RAM and depends on the protection granule.

Interleaved parity
One bit of parity is applicable to the even bits of the word, and one bit of parity is applicable to the odd bits of the word.

SECDED
Single Error Correct, Double Error Detect.

Cache protection behavior on page A8-99 indicates which protection type is applied to each RAM.

The core can progress and remain functionally correct when there is a single bit error in any RAM.

If there are multiple single bit errors in different RAMs, or within different protection granules within the same RAM, then the core also remains functionally correct.

If there is a double bit error in a single RAM within the same protection granule, then the behavior depends on the RAM:

- For RAMs with SECDED capability, the core detects and either reports or defers the error. If the error is in a cache line containing dirty data, then that data might be lost.
- For RAMs with only SED, the core does not detect a double bit error. This might cause data corruption.

If there are three or more bit errors within the same protection granule, then depending on the RAM and the position of the errors within the RAM, the core might or might not detect the errors.

The cache protection feature of the core has a minimal performance impact when no errors are present.

Table A8-1 Cache protection behavior

RAM	Protection type	Protection granule	Correction behavior
L1 instruction cache tag	2 interleaved parity bits	34 bits	The line that contains the error is invalidated from the L1 instruction cache and fetched again from the subsequent memory system.
L1 instruction cache data	2 interleaved parity bits	64 bits	The line that contains the error is invalidated from the L1 instruction cache and fetched again from the subsequent memory system.
L1 TLB small page tag	2 interleaved parity bits	78 bits	Entry invalidated, new pagewalk started to refetch it.
L1 TLB large page tag	2 interleaved parity bits	72 bits	
L1 TLB small page data	SED	51 bits	
L1 TLB large page data	SED	51 bits	
L1 data cache tag	SECDED	42 bits + 7 bits for ECC attached to the word.	The cache line that contains the error gets evicted, corrected in line, and refilled to the core.

Table A8-1 Cache protection behavior (continued)

RAM	Protection type	Protection granule	Correction behavior
L1 data cache data	SECDED	32 bits + 7 bits for ECC attached to the word.	The cache line that contains the error gets evicted, corrected in line, and refilled to the core.
L2 cache tag	SECDED SECDED	7 bits or 36 bits 7 bits or 35 bits	The cache line that contains the error gets evicted, corrected in line, and refilled to the core.
L2 cache data	SECDED	8 bits or 64 bits	Data is corrected inline.
L2 RRIP	None	-	-
L1 BTAC	None	-	-
L1 iPRED	None	-	-
L1 PRED	None	-	-
L2 SEB	SECDED	8 bits or 64 bits	Data is corrected inline.
L2 region prefetch	None	-	-

To ensure that progress is guaranteed even in case of hard error, the core returns corrected data to the core, and no cache access is required after data correction.

If an error is detected in a cache line, hardware cleans and invalidates the line from the cache.

A8.3 Uncorrected errors and data poisoning

When an error is detected, the correction mechanism is triggered. However, if the error is a 2-bit error in a RAM protected by ECC, then the error is not correctable.

The behavior on an uncorrected error depends on the type of RAM.

Uncorrected error detected in a data RAM

When an uncorrected error is detected in a data RAM, the chunk of data with the error is marked as poisoned. This poison information is then transferred with the data and stored in the cache if the data is allocated into another cache. The poisoned information is stored per 64 bits of data, except in the L1 data cache where it is stored per 32 bits of data.

Uncorrected error detected in a tag RAM

When an uncorrected error is detected in a tag RAM, either the address or coherency state of the line is not known, and the corresponding data cannot be poisoned. In this case, the line is invalidated and an error recovery interrupt is generated to notify software that data has potentially been lost.

A8.4 RAS error types

This section describes the RAS error types that are introduced by the RAS extension and supported in the Cortex-A75 core.

When a component accesses memory, an error might be detected in that memory and then be corrected, deferred, or detected but silently propagated. The following table lists the types of RAS errors that are supported in the Cortex-A75 core.

Table A8-2 RAS error types supported in the Cortex-A75 core

RAS error type	Definition
Corrected	A *Corrected Error* (CE) is reported for a single-bit ECC error on any protected RAM.
Deferred	A *Deferred Error* (DE) is reported for a double-bit ECC error that affects the data RAM on either the L1 data cache or the L2 cache.
Uncontainable	An *Uncontainable Error* (UC) is reported for a double-bit ECC error that affects the tag RAM of either the L1 data cache or the L2 cache.

A8.5 Error Synchronization Barrier

The RAS extension adds the *Error Synchronization Barrier* (ESB) instruction.

In the Cortex-A75 core, the ESB instruction allows efficient isolation of errors:

- The ESB instruction does not wait for completion of accesses that cannot generate an asynchronous external abort. For example, if all external aborts are handled synchronously or it is known that no such accesses are outstanding.
- The ESB instruction does not order accesses and does not guarantee a pipeline flush.

All uncontainable errors must be synchronized by an ESB instruction, which guarantees the following:

- All uncontainable errors that are generated before the ESB instruction have pended a *System Error Interrupts* (SEI) exception.
- If a physical SEI is pended by or was pending before the ESB instruction executes, then:
  - It is taken before completion of the ESB instruction, if the physical SEI exception is unmasked at the current Exception level.
  - The pending SEI is cleared, the SEI status is recorded in DISR/DISR_EL1, and DISR/DISR_EL1.A is set to 1 if the physical SEI exception is masked at the current Exception level. It indicates that the SEI exception was generated before the ESB instruction by instructions that occur in program order.
- If a virtual SEI is pended by or was pending before the ESB instruction executes, then:
  - It is taken before completion of the ESB instruction, if the virtual SEI exception is unmasked.
  - The pending SEI is cleared and the SEI status is recorded in VDISR/VDISR_EL2 using the information provided by software in VDFSR/VSESR_EL2, if the virtual SEI exception is masked.

After the ESB instruction, one of the following scenario happens:

- SEIs pended by errors are taken and their status is recorded in DFSR, HSR, and ESR_ELn.
- SEIs pended by errors are deferred and their status is recorded in DISR/DISR_EL1 or VDISR/VDISR_EL2.

This includes uncontainable errors that are generated by instructions, translation table walks, and instruction fetches on the same core.

Note

DISR/DISR_EL1 can only be accessed at EL1 and above. If EL2 is implemented and HCR/HCR_EL2.AMO is set to 1, then reads and writes of DISR/DISR_EL1 at Non-secure EL1 access VDISR/VDISR_EL2.

A8.6 Error recording

The component that detects an error is called a node. The Cortex-A75 core is a node that interacts with the DSU node. There is one record per node for the errors detected.

In the Cortex-A75 core, any error that is detected is reported and recorded in the error record registers described in the following table.

Table A8-3 Cortex-A75 error record registers

Register	Description
AArch64: ERXCTLR_EL1[63:0] AArch32: {ERXCTLR2[31:0], ERXCTLR[31:0]}	Selected Error Record Control Register. This register provides control on the node features.
AArch64: ERXFR_EL1[63:0] AArch32: {ERXFR2[31:0], ERXFR[31:0]}	Selected Error Record Feature Register. This register provides details on the node features.
AArch64: ERXSTATUS_EL1 AArch32: ERXSTATUS	Selected Error Record Primary Status Register. This register provides details on the type of error detected.
AArch64: ERXADDR_EL1[63:0] AArch32: {ERXADDR2[31:0], ERXADDR[31:0]}	Selected Error Record Address Register. This register provides details on the address linked to the error detected.
AArch64: ERXMISC0_EL1[63:0] AArch32: {ERXMISC1[31:0], ERXMISC0[31:0]}	Selected Error Record Miscellaneous Register 0. This register counts the errors corrected and gives the RAMs location of the last error recorded.

There are two error records provided, which can be selected with the ERRSELR/ERRSELR_EL1 register. Record 0 is private to the Cortex-A75 core, and is updated on any error in the Cortex-A75 core RAMs including L1 caches, TLB, and L2 cache. Record 1 records any error in the L3 and snoop filter RAMs and is shared between all Cortex-A75 cores in a cluster.

Detected errors and interrupts

In the Cortex-A75 core, the error recovery interrupt is controlled by ERR0CTLR.UI for UC errors, but is never enabled for DE errors.

The fault handling interrupt, controlled by ERR0CTLR.CFI and ERR0CTLR.FI, is implemented. Therefore, fault handling interrupt can be enabled for corrected error events, DE errors, and UC errors.

Error recovery and fault handling interrupts are routed using the GIC which either generates an IRQ or FIQ interrupt exception.

The fault handling interrupt is generated on the **nFAULTIRQ[0]** pin for L3 and snoop filter errors, or on the **nFAULTIRQ[n+1]** pin for core *n* L1 and L2 errors.

Uncorrected errors

Errors that cannot be corrected, and therefore might result in data corruption, also cause an abort or an interrupt signal to be asserted, alerting software to the error. The software can either attempt to recover or

can restart the system. Some errors are deferred by poisoning the data. This does not cause an abort at the time of the error, but only when the error is consumed.

- Uncorrected errors in the L1, L2, or L3 data RAMs when read by an instruction fetch, or TLB pagewalk, might result in a synchronous Data Abort or Prefetch Abort. Errors on these RAMs when read by a load might cause an SEI.
- Uncorrected errors in the L1, L2, or L3 data RAMs when the line is being evicted from a cache causes the data to be poisoned. This might be because of a natural eviction, a linefill from a higher level of cache, a cache maintenance operation, or a snoop.

Writing the error record and prioritizing errors

When a new error is detected, the node performs the following actions:

- It modifies the ERR<n>STATUS register to indicate the type of the new detected error.
- It overwrites the error record with the status of the new error if the new error has a higher priority, or keep the status of the previous error if the new error has a lower priority.
- It counts the error using the ERR<n>MISC0 register.

The following table describes the priority applied to the errors supported in the Cortex-A75 core.

Table A8-4 Rules for overwriting error records

Previous error type	New detected error type		
	CE	**DE**	**UC**
No previous error	CW	W	W
CE	CK	W	W
DE	CK	O	W
UC	CK	K	O

CK Count and keep. Count the CE error and keep the previous error status. If the new detected error makes one of the CE counters of ERR<n>MISC0 overflow, then ERR<n>STATUS.OF is set to 1.

W Overwrite. Record the new error status. If two new errors of the same type are detected simultaneously, set ERR<n>STATUS.OF to 1. Otherwise, the overwrite clears this bit.

O Overflow. Keep the previous error status and set ERR<n>STATUS.OF to 1.

K Keep. Keep the previous error status. ERR<n>STATUS.OF is unchanged.

CW Content overwrite. Count the CE error if a counter is implemented, and overwrite. ERR<n>STATUS.OF is unchanged.

Note

It is possible to count an error more than once. For example, multiple accesses can read the location with the error before the line is evicted.

Observations and constraints

The following observations should be made about ERR0STATUS and ERR0MISC registers:

- If two or more corrected memory errors occur in the same cycle and these are the first errors, then:
  - One of the errors is selected arbitrarily.
  - Its location is stored.
  - The repeat error counter is incremented only once.
- If two or more corrected memory errors occur in the same cycle and these are not the first errors, then:

  - One of the errors is selected arbitrarily.
  - One of the two counters is incremented only once. The location of the selected error and the location of the last recorded error define which counter is incremented.
- If a new corrected error arrives while the ERR0STATUS.V bit is set, and a corrected error is already recorded, the way, index, and level information stored in ERR0MISC0 is not updated, but the other error counter or the repeat error counter is incremented depending on the location currently stored.
- If two or more corrected memory errors from different RAMs that do not match the level, way and index information in this register when the ERR0STATUS.V bit is set, occur in the same cycle, the Other error count field is only incremented once.
- This register is not reset on a Warm reset. It is instead reset on a Cold reset.
- ERR0MISC0[31:0], along with ERR0STATUS.SERR, stores the location of the last error that overwrites the record. This location is used to select which of the two counters is incremented on a CE error. When an overwriting error is detected, this location is updated with the location of the new error. Therefore, if an error with a higher priority than a CE is recorded, for example a DE or UC error, then the count split is not representative of the location stored in ERR0MISC0 and in ERR0STATUS.SERR.

A8.7 Error injection

To support testing of error handling software, the Cortex-A75 core can fake errors in the error detection logic.

The following table describes all the possible types of error that the core can encounter and therefore fake.

Table A8-5 Errors injected in the Cortex-A75 core

Error type	Description
Corrected errors	A CE is generated for a single ECC error on L1 data caches or L2 caches, either on data RAM or tag RAM.
Deferred errors	A DE is generated for a double ECC error on L1 data caches or L2 caches, but only on data RAM.
Uncontainable errors	A UC is generated for a double ECC error on L1 data caches or L2 caches, but only on tag RAM.

The following table describes the registers that handle error injection in the Cortex-A75 core.

Table A8-6 Error injection registers

Register name	Description
ERR0PFGFR	The ERR Pseudo Fault Generation Feature register defines which errors can be injected.
ERR0PFGCTLR	The ERR Pseudo Fault Generation Control register controls the errors that are injected.
ERR0PFGCDNR	The ERR Pseudo Fault Generation Count Down register controls the fault injection timing.

Note

This mechanism simulates the corruption of any RAM but the data is not actually corrupted.

See also:

- *B3.9 ERR0PFGFR, Error Pseudo Fault Generation Feature Register* on page B3-451.
- *B3.8 ERR0PFGCTLR, Error Pseudo Fault Generation Control Register* on page B3-449.
- *B3.7 ERR0PFGCDNR, Error Pseudo Fault Generation Count Down Register* on page B3-448.

Note

The fake error to be injected is only generated at the next event which can report such error in real case, that is at the next RAM access that is supposed to generate such error. It means that the fake error might not be directly generated at the end of the countdown.

Chapter A9
Generic Interrupt Controller CPU Interface

This chapter describes the Cortex-A75 core implementation of the Arm *Generic Interrupt Controller* (GIC) CPU interface.

It contains the following sections:

A9.1 About the Generic Interrupt Controller CPU Interface

The GIC CPU Interface, when integrated with an external distributor component, is a resource for supporting and managing interrupts in a cluster system.

The GIC CPU interface hosts registers to mask, identify, and control states of interrupts forwarded to that core. There is a separate GIC CPU interface for each core in the system.

The Cortex-A75 core implements the GIC CPU interface as described in the *Arm® Generic Interrupt Controller Architecture Specification*. This interfaces with an external GICv3 or GICv4 interrupt distributor component within the system.

Note

This chapter describes only features that are specific to the Cortex-A75 core implementation. Additional information specific to the DSU can be found in *Arm® DynamIQ™ Shared Unit Technical Reference Manual*.

The GICv4 architecture supports:

- Two security states.
- Interrupt virtualization.
- *Software-generated Interrupts* (SGIs).
- Message Based Interrupts.
- System register access for the CPU interface.
- Interrupt masking and prioritization.
- Cluster environments, including systems that contain more than eight cores.
- Wake-up events in power management environments.

The GIC includes interrupt grouping functionality that supports:

- Configuring each interrupt to belong to an interrupt group.
- Signaling Group 1 interrupts to the target core using either the IRQ or the FIQ exception request.
- Signaling Group 0 interrupts to the target core using the FIQ exception request only.
- A unified scheme for handling the priority of Group 0 and Group 1 interrupts.

This chapter describes only features that are specific to the Cortex-A75 core implementation.

A9.2 Bypassing the CPU Interface

The GIC CPU Interface is always implemented within the Cortex-A75 core.

However, you can disable it if you assert the **GICCDISABLE** signal HIGH at reset. If the GIC is enabled, the input pins **nVIRQ** and **nVFIQ** must be tied off to HIGH. This is because the internal GIC CPU interface generates the virtual interrupt signals to the cores. The **nIRQ** and **nFIQ** signals are controlled by software, therefore there is no requirement to tie them HIGH. If you disable the GIC CPU interface, the input pins **nVIRQ** and **nVFIQ** can be driven by an external GIC in the SoC.

If the Cortex-A75 core is not integrated with an external GICv3 or GICv4 distributor component in the system, then you can disable the GIC CPU Interface by asserting the **GICCDISABLE** signal HIGH at reset.

GIC system register access generates UNDEFINED instruction exceptions when the **GICCDISABLE** signal is HIGH.

Chapter A10
Advanced SIMD and Floating-point support

This chapter describes the Advanced SIMD and floating-point features and registers in the Cortex-A75 core. The unit in charge of handling the Advanced SIMD and floating-point features is also referred to as data engine in this manual.

It contains the following sections:

A10.1 About the Advanced SIMD and floating-point support

The Cortex-A75 core supports the Advanced SIMD and scalar floating-point instructions in the A64 instruction set and the Advanced SIMD and floating-point instructions in the A32 and T32 instruction sets.

The Cortex-A75 floating-point implementation:

- Does not generate floating-point exceptions.
- Implements all scalar operations in hardware with support for all combinations of:
  - Rounding modes.
  - Flush-to-zero.
  - Default *Not a Number* (NaN) modes.

The Armv8 architecture does not define a separate version number for its Advanced SIMD and floating-point support in the AArch64 execution state because the instructions are always implicitly present.

A10.2 Accessing the feature identification registers

Software can identify the Advanced SIMD and floating-point features using the feature identification registers in the AArch64 and AArch32 Execution states.

You can access the feature identification registers in the AArch64 Execution state using the `MRS` instruction, for example:

```
MRS <Xt>, ID_AA64PFR0_EL1 ; Read ID_AA64PFR0_EL1 into Xt
MRS <Xt>, MVFR0_EL1       ; Read MVFR0_EL1 into Xt
MRS <Xt>, MVFR1_EL1       ; Read MVFR1_EL1 into Xt
MRS <Xt>, MVFR2_EL1       ; Read MVFR2_EL1 into Xt
```

Table A10-1 AArch64 Advanced SIMD and scalar floating-point feature identification registers

AArch64 name	Description
ID_AA64PFR0_EL1	. *B2.61 ID_AA64PFR0_EL1, AArch64 Processor Feature Register 0, EL1* on page B2-371.
MVFR0_EL1	See *B5.2.3 MVFR0_EL1, Media and VFP Feature Register 0, EL1* on page B5-524.
MVFR1_EL1	See *B5.2.4 MVFR1_EL1, Media and VFP Feature Register 1, EL1* on page B5-525.
MVFR2_EL1	See *B5.2.5 MVFR2_EL1, Media and VFP Feature Register 2, EL1* on page B5-527.

You can access the feature identification registers in the AArch32 Execution state using the `VMRS` instruction, for example:

```
VMRS <Rt>, FPSID ; Read FPSID into Rt
VMRS <Rt>, MVFR0 ; Read MVFR0 into Rt
VMRS <Rt>, MVFR1 ; Read MFFR1 into Rt
VMRS <Rt>, MVFR2 ; Read MVFR2 into Rt
```

Table A10-2 AArch32 Advanced SIMD and scalar floating-point feature identification registers

AArch32 name	Description
FPSID	See *B5.4.1 FPSID, Floating-Point System ID Register* on page B5-531.
MVFR0	See *B5.4.3 MVFR0, Media and VFP Feature Register 0* on page B5-535.
MVFR1	See *B5.4.4 MVFR1, Media and VFP Feature Register 1* on page B5-536.
MVFR2	See *B5.4.5 MVFR2, Media and VFP Feature Register 2* on page B5-538.

Part B
Register descriptions

Chapter B1
AArch32 system registers

This chapter describes the system registers in the AArch32 state.

It contains the following sections:

B1.1 AArch32 registers

This chapter provides information about AArch32 system registers with implementation defined bit fields and implementation defined registers associated with the core.

The chapter provides implementation specific information, for a complete description of the registers, see the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.The chapter is presented as follows:

AArch32 architectural system register summary
This section identifies the AArch32 architecturally defined registers implemented in the Cortex-A75 core.

The first table identifies the registers that have implementation defined bit fields. The register descriptions for these registers only contain information about the implementation defined bits.

The second table identifies the other architecturally defined registers that are implemented in the Cortex-A75 core. These registers are described in the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

AArch32 implementation defined register summary
This section identifies the AArch32 registers implemented in the Cortex-A75 core that are implementation defined.

AArch32 registers by functional group
This section groups the implementation defined registers and architectural system registers with implementation defined bit fields, as identified previously, by function. It also provides reset details for key register types.

Register descriptions
The remainder of the chapter provides register descriptions of the implementation defined registers and architectural system registers with implementation defined bit fields, as identified previously. These are listed in alphabetic order.

B1.2 AArch32 architectural system register summary

This section identifies the AArch32 architectural system registers implemented in the Cortex-A75 core.

The section contains two tables:

Registers with implementation defined bit fields

This table identifies the architecturally defined registers in the Cortex-A75 core that have IMPLEMENTATION DEFINED bit fields. The register descriptions for these registers only contain information about the implementation defined features.

See *Table B1-1 Registers with implementation defined bit fields* on page B1-123.

Other architecturally defined registers

This table identifies the other architecturally defined registers that are implemented in the Cortex-A75 core. These registers are described in the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

See *Table B1-2 Other architecturally defined registers* on page B1-126.

Registers with implementation defined bit fields

For all the registers listed in the following table, coproc==`0b1111`.

Table B1-1 Registers with implementation defined bit fields

Name	CRn	Opc1	CRm	Opc2	Width	Description
ACTLR	c1	0	c0	1	32	*B1.5 ACTLR, Auxiliary Control Register* on page B1-136
ACTLR2	c1	0	c0	3	32	*B1.6 ACTLR2, Auxiliary Control Register 2* on page B1-138
AIDR	c0	1	c0	7	32	*B1.8 AIDR, Auxiliary ID Register* on page B1-140
ADFSR	c5	0	c1	0	32	*B1.7 ADFSR, Auxiliary Data Fault Status Register* on page B1-139
AIFSR	c5	0	c1	1	32	*B1.9 AIFSR, Auxiliary Instruction Fault Status Register* on page B1-141
AMAIR0	c10	0	c3	0	32	*B1.10 AMAIR0, Auxiliary Memory Attribute Indirection Register 0* on page B1-142
AMAIR1	c10	0	c3	1	32	*B1.11 AMAIR1, Auxiliary Memory Attribute Indirection Register 1* on page B1-143
CCSIDR	c0	1	c0	0	32	*B1.12 CCSIDR, Cache Size ID Register* on page B1-144
CLIDR	c0	1	c0	1	32	*B1.13 CLIDR, Cache Level ID Register* on page B1-146
CPACR	c1	0	c0	2	32	*B1.14 CPACR, Architectural Feature Access Control Register* on page B1-148
CSSELR	c0	2	c0	0	32	*B1.24 CSSELR, Cache Size Selection Register* on page B1-168
CTR	c0	0	c0	1	32	*B1.25 CTR, Cache Type Register* on page B1-169
DFSR	c5	0	c0	0	32	*B1.26 DFSR, Data Fault Status Register* on page B1-171
DISR	c12	0	c1	1	32	*B1.27 DISR, Deferred Interrupt Status Register* on page B1-173
ERRIDR	c5	0	c3	0	32	*B1.28 ERRIDR, Error ID Register* on page B1-176
ERRSELR	c5	0	c3	1	32	*B1.29 ERRSELR, Error Record Select Register* on page B1-177
ERXADDR	c5	0	c4	3	32	*B1.30 ERXADDR, Selected Error Record Address Register* on page B1-178

Table B1-1 Registers with implementation defined bit fields (continued)

Name	CRn	Opc1	CRm	Opc2	Width	Description
ERXADDR2	c5	0	c4	7	32	*B1.31 ERXADDR2, Selected Error Record Address Register 2* on page B1-179
ERXCTLR	c5	0	c4	1	32	*B1.32 ERXCTLR, Selected Error Record Control Register* on page B1-180
ERXCTLR2	c5	0	c4	5	32	*B1.33 ERXCTLR2, Selected Error Record Control Register 2* on page B1-181
ERXFR	c5	0	c4	0	32	*B1.34 ERXFR, Selected Error Record Feature Register* on page B1-182
ERXFR2	c5	0	c4	4	32	*B1.35 ERXFR2, Selected Error Record Feature Register 2* on page B1-183
ERXMISC0	c5	0	c5	0	32	*B1.36 ERXMISC0, Selected Error Miscellaneous Register 0* on page B1-184
ERXMISC1	c5	0	c5	1	32	*B1.37 ERXMISC1, Selected Error Miscellaneous Register 1* on page B1-185
ERXMISC2	c5	0	c5	4	32	*B1.38 ERXMISC2, Selected Error Record Miscellaneous Register 2* on page B1-186
ERXMISC3	c5	0	c5	5	32	*B1.39 ERXMISC3, Selected Error Record Miscellaneous Register 3* on page B1-187
ERXSTATUS	c5	0	c4	2	32	*B1.43 ERXSTATUS, Selected Error Record Primary Status Register* on page B1-193
FCSEIDR	c13	0	c0	0	32	*B1.44 FCSEIDR, FCSE Process ID Register* on page B1-194
FPSID	c5	4	c3	1	32	*B5.4.1 FPSID, Floating-Point System ID Register* on page B5-531.
HACR	c1	4	c1	7	32	*B1.45 HACR, Hyp Auxiliary Configuration Register* on page B1-195
HACTLR	c1	4	c0	1	32	*B1.46 HACTLR, Hyp Auxiliary Control Register* on page B1-196
HACTLR2	c1	4	c0	3	32	*B1.47 HACTLR2, Hyp Auxiliary Control Register 2* on page B1-198
HADFSR	c5	4	c1	0	32	*B1.48 HADFSR, Hyp Auxiliary Data Fault Status Syndrome Register* on page B1-199
HAIFSR	c5	4	c1	1	32	*B1.49 HAIFSR, Hyp Auxiliary Instruction Fault Status Syndrome Register* on page B1-200
HAMAIR0	c10	4	c3	0	32	*B1.50 HAMAIR0, Hyp Auxiliary Memory Attribute Indirection Register 0* on page B1-201
HAMAIR1	c10	4	c3	1	32	*B1.51 HAMAIR1, Hyp Auxiliary Memory Attribute Indirection Register 1* on page B1-202
HCR	c1	4	c1	0	32	*B1.52 HCR, Hyp Configuration Register* on page B1-203
HCR2	c1	4	c1	4	32	*B1.53 HCR2, Hyp Configuration Register 2* on page B1-205
HSR	c5	4	c2	0	32	*B1.55 HSR, Hyp Syndrome Register* on page B1-208
ID_AFR0	c0	0	c1	3	32	*B1.57 ID_AFR0, Auxiliary Feature Register 0* on page B1-211
ID_DFR0	c0	0	c1	2	32	*B1.58 ID_DFR0, Debug Feature Register 0* on page B1-212
ID_ISAR0	c0	0	c2	0	32	*B1.59 ID_ISAR0, Instruction Set Attribute Register 0* on page B1-214

Table B1-1 Registers with implementation defined bit fields (continued)

Name	CRn	Opc1	CRm	Opc2	Width	Description
ID_ISAR1	c0	0	c2	1	32	*B1.60 ID_ISAR1, Instruction Set Attribute Register 1* on page B1-216
ID_ISAR2	c0	0	c2	2	32	*B1.61 ID_ISAR2, Instruction Set Attribute Register 2* on page B1-218
ID_ISAR3	c0	0	c2	3	32	*B1.62 ID_ISAR3, Instruction Set Attribute Register 3* on page B1-220
ID_ISAR4	c0	0	c2	4	32	*B1.63 ID_ISAR4, Instruction Set Attribute Register 4* on page B1-222
ID_ISAR5	c0	0	c2	5	32	*B1.64 ID_ISAR5, Instruction Set Attribute Register 5* on page B1-224
ID_ISAR6	c0	0	c2	7	32	*B1.65 ID_ISAR6, Instruction Set Attribute Register 6* on page B1-226
ID_MMFR0	c0	0	c1	4	32	*B1.66 ID_MMFR0, Memory Model Feature Register 0* on page B1-227
ID_MMFR1	c0	0	c1	5	32	*B1.67 ID_MMFR1, Memory Model Feature Register 1* on page B1-229
ID_MMFR2	c0	0	c1	6	32	*B1.68 ID_MMFR2, Memory Model Feature Register 2* on page B1-231
ID_MMFR3	c0	0	c1	7	32	*B1.69 ID_MMFR3, Memory Model Feature Register 3* on page B1-233
ID_MMFR4	c0	0	c2	6	32	*B1.70 ID_MMFR4, Memory Model Feature Register 4* on page B1-235
ID_PFR0	c0	0	c1	0	32	*B1.71 ID_PFR0, Processor Feature Register 0* on page B1-237
ID_PFR1	c0	0	c1	1	32	*B1.72 ID_PFR1, Processor Feature Register 1* on page B1-238
IFSR	c5	0	c0	1	32	*B1.73 IFSR, Instruction Fault Status Register* on page B1-240
MIDR	c0	0	c0	0, 4, or 7	32	*B1.74 MIDR, Main ID Register* on page B1-242
MPIDR	c0	0	c0	5	32	*B1.75 MPIDR, Multiprocessor Affinity Register* on page B1-243
PAR (32 bits access)	c7	0	c4	0	32	*B1.78 PAR, Physical Address Register* on page B1-247
PAR (64 bits access)	-	0	c7	-	64	*B1.78 PAR, Physical Address Register* on page B1-247
REVIDR	c0	0	c0	6	32	*B1.79 REVIDR, Revision ID Register* on page B1-249
SCR	c1	0	c1	0	32	*B1.81 SCR, Secure Configuration Register* on page B1-251
SCTLR	c1	0	c0	0	32	*B1.82 SCTLR, System Control Register* on page B1-252
SDCR	c1	0	c3	1	32	*B1.83 SDCR, Secure Debug Control Register* on page B1-255
TTBCR	c2	0	c0	2	32	*B1.84 TTBCR, Translation Table Base Control Register* on page B1-257
TTBR0 (32 bits access)	c2	0	c0	0	32	*B1.85 TTBR0, Translation Table Base Register 0* on page B1-258
TTBR0 (64 bits access)	-	0	c2	-	64	*B1.85 TTBR0, Translation Table Base Register 0* on page B1-258

Table B1-1 Registers with implementation defined bit fields (continued)

Name	CRn	Opc1	CRm	Opc2	Width	Description
TTBR1 (32 bits access)	c2	0	c0	1	32	*B1.86 TTBR1, Translation Table Base Register 1* on page B1-260
TTBR1 (64 bits access)	-	1	c2	-	64	*B1.86 TTBR1, Translation Table Base Register 1* on page B1-260
VDFSR	c5	4	c2	3	32	*B1.88 VDFSR, Virtual SError Exception Syndrome Register* on page B1-263
VDISR	c12	4	c1	1	32	*B1.89 VDISR, Virtual Deferred Interrupt Status Register* on page B1-264
VMPIDR	c0	4	c0	5	32	*B1.90 VMPIDR, Virtualization Multiprocessor ID Register* on page B1-267
VPIDR	c0	4	c0	0	32	*B1.91 VPIDR, Virtualization Processor ID Register* on page B1-268
VTCR	c2	4	c1	2	32	*B1.92 VTCR, Virtualization Translation Control Register* on page B1-269
VTTBR	-	6	c2	-	64	*B1.93 VTTBR, Virtualization Translation Table Base Register* on page B1-272

Other architecturally defined registers

For the registers listed in the following table, coproc==`0b1111`, except for:

- Jazelle ID Register.
- Jazelle Main Configuration Register.
- Jazelle OS Control Register.

For these registers, coproc==`0b1110`.

Table B1-2 Other architecturally defined registers

Name	CRn	Opc1	CRm	Opc2	Width	description
CPACR	c1	0	c0	2	32	Architectural Feature Access Control Register
CNTFRQ	c14	0	c0	0	32	Timer Clock Ticks per Second
CNTHCTL	c14	4	c1	0	32	Timer Hyp Control register
CNTHP_CTL	c14	4	c2	1	32	Counter-timer Hyp Physical Timer Control register
CNTHP_CVAL	-	6	c14	-	64	Counter-timer Hyp Physical CompareValue register
CNTHP_TVAL	c14	4	c2	0	32	Counter-timer Hyp Physical Timer TimerValue register
CNTKCTL	c14	0	c1	0	32	Counter-timer Kernel Control register
CNTP_CTL	c14	0	c2	1	32	Counter-timer Physical Timer Control register
CNTP_CVAL	-	2	c14	-	64	Counter-timer Physical Timer CompareValue register
CNTP_TVAL	c14	0	c2	0	32	Counter-timer Physical Timer TimerValue register
CNTPCT	-	0	c14	-	64	Counter-timer Physical Count register
CNTV_CTL	c14	0	c3	1	32	Counter-timer Virtual Timer Control register
CNTV_CVAL	-	3	c14	-	64	Counter-timer Virtual Timer CompareValue register
CNTV_TVAL	c14	0	c3	0	32	Counter-timer Virtual Timer TimerValue register

Table B1-2 Other architecturally defined registers (continued)

Name	CRn	Opc1	CRm	Opc2	Width	description
CNTVCT	-	1	c14	-	64	Counter-timer Virtual Count register
CNTVOFF	-	4	c14	-	64	Counter-timer Virtual Offset register
CONTEXTIDR	c13	0	c0	1	32	Context ID Register
DACR	c3	0	c0	0	32	Domain Access Control Register
DFAR	c6	0	c0	0	32	Data Fault Address Register
DLR	c4	3	c5	1	32	Debug Link Register
DSPSR	c4	3	c5	0	32	Debug Saved Program Status Register
FPEXC	c5	4	c3	0	32	Floating-point Exception Control register
HCPTR	c1	4	c1	2	32	Hypervisor Coprocessor Trap Register
HDCR	c1	4	c1	1	32	Hypervisor Debug Control Register
HDFAR	c6	4	c0	0	32	Hypervisor Data Fault Address
HIFAR	c6	4	c0	2	32	Hypervisor Instruction Fault Address
HMAIR0	c10	4	c2	0	32	Hypervisor Memory Attribute Indirection Register 0
HMAIR1	c10	4	c2	1	32	Hypervisor Memory Attribute Indirection Register 1
HPFAR	c6	4	c0	4	32	Hypervisor IPA Fault Address
HTCR	c2	4	c0	2	32	Hypervisor Translation Control Register
HTPIDR	c13	4	c0	2	32	Hypervisor Software Thread ID Register
HTTBR	-	4	c2	-	64	Hypervisor Translation Table Base Register
HVBAR	c12	4	c0	0	32	Hypervisor Vector Base Address
IFAR	c6	0	c0	2	32	Instruction Fault Address Register
ISR	c12	0	c1	0	32	Interrupt Status Register
JIDR	c0	7	c0	0	32	Jazelle ID Register
JMCR	c2	7	c0	0	32	Jazelle Main Configuration Register
JOSCR	c1	7	c0	0	32	Jazelle OS Control Register
MAIR0	c10	0	c2	1	32	Memory Attribute Indirection Register 0
MAIR1	c10	0	c2	1	32	Memory Attribute Indirection Register 1
MVBAR	c12	0	c0	1	32	Monitor Vector Base Address Register
MVFR0	c0	2	c3	0	32	Media and VFP Feature Register 0
MVFR1	c0	2	c3	1	32	Media and VFP Feature Register 1
MVFR2	c0	2	c3	2	32	Media and VFP Feature Register 2
NMRR	c10	0	c2	1	32	Normal Memory Remap Register
PRRR	c10	0	c2	0	32	Primary Region Remap Register
RMR	c12	0	c0	2	32	Reset Management Register
SDER	c1	0	c1	1	32	Secure Debug Enable Register
TCMTR	c0	0	c0	2	32	TCM Type Register

Table B1-2 Other architecturally defined registers (continued)

Name	CRn	Opc1	CRm	Opc2	Width	description
TLBTR	c0	0	c0	3	32	TLB Type Register
TPIDRPRW	c13	0	c0	4	32	Privileged Only Thread ID Register
TPIDRURO	c13	0	c0	3	32	User Read Only Thread ID Register
TPIDRURW	c13	0	c0	2	32	User Read/Write Thread ID Register
VBAR	c12	0	c0	0	32	Vector Base Address Register

B1.3 AArch32 implementation defined register summary

This section identifies the AArch32 registers implemented in the Cortex-A75 core that are implementation defined. The list of registers is sorted by opcode.

The registers that are implemented but are architecturally defined are described in the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

Table B1-3 Cortex-A75 AArch32 implementation defined registers

Name	Copro	CRn	Opc1	CRm	Opc2	Width	Description
CPUACTLR	cp15	-	0	c15	-	64	*B1.15 CPUACTLR, CPU Auxiliary Control Register* on page B1-149
CPUACTLR2	cp15	-	1	c15	-	64	*B1.16 CPUACTLR2, CPU Auxiliary Control Register 2* on page B1-151
CPUCFR	cp15	c15	0	c0	0	32	*B1.17 CPUCFR, CPU Configuration Register* on page B1-153
CPUECTLR	cp15	-	4	c15	-	64	*B1.18 CPUECTLR, CPU Extended Control Register* on page B1-155
CPUPCR	cp15	-	8	c15	-	64	*B1.19 CPUPCR, CPU Private Control Register* on page B1-158
CPUPMR	cp15	-	10	c15	-	64	*B1.20 CPUPMR, CPU Private Mask Register* on page B1-160
CPUPOR	cp15	-	9	c15	-	64	*B1.21 CPUPOR, CPU Private Operation Register* on page B1-162
CPUPSELR	cp15	c15	6	c8	0	32	*B1.22 CPUPSELR, CPU Private Selection Register* on page B1-164
CPUPWRCTLR	cp15	c15	0	c2	7	32	*B1.23 CPUPWRCTLR, CPU Power Control Register* on page B1-166
ERR0PFGFR [e]	N/A	N/A	N/A	N/A	N/A	32	*B3.9 ERR0PFGFR, Error Pseudo Fault Generation Feature Register* on page B3-451
ERR0PFGCTLR [e]	N/A	N/A	N/A	N/A	N/A	32	*B3.8 ERR0PFGCTLR, Error Pseudo Fault Generation Control Register* on page B3-449
ERR0PFGCDNR [e]	N/A	N/A	N/A	N/A	N/A	32	*B3.7 ERR0PFGCDNR, Error Pseudo Fault Generation Count Down Register* on page B3-448
ERXPFGCDNR	cp15	c15	0	c2	2	32	*B1.40 ERXPFGCDNR, Selected Error Pseudo Fault Generation Count Down Register* on page B1-188
ERXPFGCTLR	cp15	c15	0	c2	1	32	*B1.41 ERXPFGCTLR, Selected Error Pseudo Fault Generation Control Register* on page B1-190
ERXPFGFR	cp15	c15	0	c2	0	32	*B1.42 ERXPFGFR, Selected Pseudo Fault Generation Feature Register* on page B1-192

The following table shows the 32-bit wide implementation defined Cluster registers. Details of these registers can be found in *Arm® DynamIQ™ Shared Unit Technical Reference Manual*

Table B1-4 Cluster registers

Name	Copro	CRn	Opc1	CRm	Opc2	Width	Description
CLUSTERCFR	cp15	c15	0	c3	0	32-bit	Cluster configuration register.
CLUSTERIDR	cp15	c15	0	c3	1	32-bit	Cluster main revision ID.
CLUSTEREVIDR	cp15	c15	0	c3	2	32-bit	Cluster ECO ID.

[e] There is no direct access to ERR0* registers using `MCR` and `MRC`.

Table B1-4 Cluster registers (continued)

Name	Copro	CRn	Opc1	CRm	Opc2	Width	Description
CLUSTERACTLR	cp15	c15	0	c3	3	32-bit	Cluster auxiliary control register.
CLUSTERECTLR	cp15	c15	0	c3	4	32-bit	Cluster extended control register.
CLUSTERPWRCTLR	cp15	c15	0	c3	5	32-bit	Cluster power control register.
CLUSTERPWRDN	cp15	c15	0	c3	6	32-bit	Cluster power down register.
CLUSTERPWRSTAT	cp15	c15	0	c3	7	32-bit	Cluster power status register.
CLUSTERTHREADSID	cp15	c15	0	c4	0	32-bit	Cluster thread scheme ID register.
CLUSTERACPSID	cp15	c15	0	c4	1	32-bit	Cluster ACP scheme ID register.
CLUSTERSTASHSID	cp15	c15	0	c4	2	32-bit	Cluster stash scheme ID register.
CLUSTERPARTCR	cp15	c15	0	c4	3	32-bit	Cluster partition control register.
CLUSTERBUSQOS	cp15	c15	0	c4	4	32-bit	Cluster bus QoS control register.
CLUSTERL3HIT	cp15	c15	0	c4	5	32-bit	Cluster L3 hit counter register.
CLUSTERL3MISS	cp15	c15	0	c4	6	32-bit	Cluster L3 miss counter register.
CLUSTERTHREADSIDOVR	cp15	c15	0	c4	7	32-bit	Cluster thread scheme ID override register.
CLUSTERPM*	cp15	c15	0 or 6	c5-c6	0-7	32-bit or 64-bit	Cluster PMU registers

B1.4 AArch32 registers by functional group

This section identifies the AArch32 registers by their functional groups and applies to the registers in the core that are implementation defined or have micro-architectural bit fields.

Reset values are provided for these registers.

Identification registers

Name	Type	Reset	Description
AIDR	RO	`0x00000000`	*B1.8 AIDR, Auxiliary ID Register* on page B1-140
CCSIDR	RO	-	*B1.12 CCSIDR, Cache Size ID Register* on page B1-144
CLIDR	RO	• `0xC3000123` if L3 cache is present. • `0x82000023` if no L3 cache is present.	*B1.13 CLIDR, Cache Level ID Register* on page B1-146
CSSELR	RW	UNK	*B1.24 CSSELR, Cache Size Selection Register* on page B1-168
CTR	RO	`0x84448004`	*B1.25 CTR, Cache Type Register* on page B1-169
ID_AFR0	RO	`0x00000000`	*B1.57 ID_AFR0, Auxiliary Feature Register 0* on page B1-211
ID_DFR0	RO	`0x04010088`	*B1.58 ID_DFR0, Debug Feature Register 0* on page B1-212 Bits [19:16] are `0x1` if ETM is implemented, and `0x0` otherwise.
ID_ISAR0	RO	`0x02101110`	*B1.59 ID_ISAR0, Instruction Set Attribute Register 0* on page B1-214
ID_ISAR1	RO	`0x13112111`	*B1.60 ID_ISAR1, Instruction Set Attribute Register 1* on page B1-216
ID_ISAR2	RO	`0x21232042`	*B1.61 ID_ISAR2, Instruction Set Attribute Register 2* on page B1-218
ID_ISAR3	RO	`0x01112131`	*B1.62 ID_ISAR3, Instruction Set Attribute Register 3* on page B1-220
ID_ISAR4	RO	`0x00011142`	*B1.63 ID_ISAR4, Instruction Set Attribute Register 4* on page B1-222
ID_ISAR5	RO	`0x00011121`	*B1.64 ID_ISAR5, Instruction Set Attribute Register 5* on page B1-224 ID_ISAR5 has the value `0x00010001` if the Cryptographic Extension is not implemented and enabled.
ID_ISAR6	RO	`0x00000010`	*B1.65 ID_ISAR6, Instruction Set Attribute Register 6* on page B1-226
ID_MMFR0	RO	`0x10201105`	*B1.66 ID_MMFR0, Memory Model Feature Register 0* on page B1-227
ID_MMFR1	RO	`0x40000000`	*B1.67 ID_MMFR1, Memory Model Feature Register 1* on page B1-229
ID_MMFR2	RO	`0x01260000`	*B1.68 ID_MMFR2, Memory Model Feature Register 2* on page B1-231
ID_MMFR3	RO	`0x02102211`	*B1.69 ID_MMFR3, Memory Model Feature Register 3* on page B1-233
ID_MMFR4	RO	`0x00021110`	*B1.70 ID_MMFR4, Memory Model Feature Register 4* on page B1-235
ID_PFR0	RO	`0x00000131`	*B1.71 ID_PFR0, Processor Feature Register 0* on page B1-237
ID_PFR1	RO	`0x10011011`	*B1.72 ID_PFR1, Processor Feature Register 1* on page B1-238 Bits [31:28] are `0x1` if the GIC CPU interface is implemented and enabled, and `0x0` otherwise.
MIDR	RO	`0x412FD0A1`	*B1.74 MIDR, Main ID Register* on page B1-242
MPIDR	RO	-	*B1.75 MPIDR, Multiprocessor Affinity Register* on page B1-243
REVIDR	RO	`0x00000000`	*B1.79 REVIDR, Revision ID Register* on page B1-249

(continued)

Name	Type	Reset	Description
VMPIDR	RW	-	*B1.90 VMPIDR, Virtualization Multiprocessor ID Register* on page B1-267 The reset value is the value of MPIDR.
VPIDR	RW	`0x412FD0A1`	*B1.91 VPIDR, Virtualization Processor ID Register* on page B1-268

Other system control registers

Name	Type	Description
ACTLR	RW	*B1.5 ACTLR, Auxiliary Control Register* on page B1-136
ACTLR2	RW	*B1.6 ACTLR2, Auxiliary Control Register 2* on page B1-138
HACTLR	RW	*B1.46 HACTLR, Hyp Auxiliary Control Register* on page B1-196
HACTLR2	RW	*B1.47 HACTLR2, Hyp Auxiliary Control Register 2* on page B1-198
HSCTLR	RW	*B1.54 HSCTLR, Hyp System Control Register* on page B1-206
SCTLR	RW	*B1.82 SCTLR, System Control Register* on page B1-252

RAS registers

Name	Type	Description
DISR	RW	*B1.27 DISR, Deferred Interrupt Status Register* on page B1-173
ERRIDR	RO	*B1.28 ERRIDR, Error ID Register* on page B1-176
ERRSELR	RW	*B1.29 ERRSELR, Error Record Select Register* on page B1-177
ERXADDR	RW	*B1.30 ERXADDR, Selected Error Record Address Register* on page B1-178
ERXADDR2	RW	*B1.31 ERXADDR2, Selected Error Record Address Register 2* on page B1-179
ERXCTLR	RW	*B1.32 ERXCTLR, Selected Error Record Control Register* on page B1-180
ERXCTLR2	RW	*B1.33 ERXCTLR2, Selected Error Record Control Register 2* on page B1-181
ERXFR	RO	*B1.34 ERXFR, Selected Error Record Feature Register* on page B1-182
ERXFR2	RO	*B1.35 ERXFR2, Selected Error Record Feature Register 2* on page B1-183
ERXMISC0	RW	*B1.36 ERXMISC0, Selected Error Miscellaneous Register 0* on page B1-184
ERXMISC1	RW	*B1.37 ERXMISC1, Selected Error Miscellaneous Register 1* on page B1-185
ERXMISC2	RW	*B1.38 ERXMISC2, Selected Error Record Miscellaneous Register 2* on page B1-186
ERXMISC3	RW	*B1.39 ERXMISC3, Selected Error Record Miscellaneous Register 3* on page B1-187
ERXPFGCDNR	RW	*B1.40 ERXPFGCDNR, Selected Error Pseudo Fault Generation Count Down Register* on page B1-188
ERXPFGCTLR	RW	*B1.41 ERXPFGCTLR, Selected Error Pseudo Fault Generation Control Register* on page B1-190
ERXPFGFR	RO	*B1.42 ERXPFGFR, Selected Pseudo Fault Generation Feature Register* on page B1-192
ERXSTATUS	RW	*B1.43 ERXSTATUS, Selected Error Record Primary Status Register* on page B1-193
HCR2	RW	*B1.53 HCR2, Hyp Configuration Register 2* on page B1-205
VDFSR	RW	*B1.88 VDFSR, Virtual SError Exception Syndrome Register* on page B1-263
VDISR	RW	*B1.89 VDISR, Virtual Deferred Interrupt Status Register* on page B1-264

Virtual Memory control registers

Name	Type	Description
AMAIR0	RW	*B1.10 AMAIR0, Auxiliary Memory Attribute Indirection Register 0* on page B1-142
AMAIR1	RW	*B1.11 AMAIR1, Auxiliary Memory Attribute Indirection Register 1* on page B1-143
HAMAIR0	RW	*B1.50 HAMAIR0, Hyp Auxiliary Memory Attribute Indirection Register 0* on page B1-201
HAMAIR1	RW	*B1.51 HAMAIR1, Hyp Auxiliary Memory Attribute Indirection Register 1* on page B1-202
HTTBR	RW	*B1.56 HTTBR, Hyp Translation Table Base Register* on page B1-210
SCTLR	RW	*B1.82 SCTLR, System Control Register* on page B1-252
TTBCR	RW	*B1.84 TTBCR, Translation Table Base Control Register* on page B1-257
TTBR0	RW	*B1.85 TTBR0, Translation Table Base Register 0* on page B1-258
TTBR1	RW	*B1.86 TTBR1, Translation Table Base Register 1* on page B1-260
VTCR	RW	*B1.92 VTCR, Virtualization Translation Control Register* on page B1-269
VTTBR	RW	*B1.93 VTTBR, Virtualization Translation Table Base Register* on page B1-272

Virtualization registers

Name	Type	Description
HACR	RW	*B1.45 HACR, Hyp Auxiliary Configuration Register* on page B1-195
HACTLR	RW	*B1.46 HACTLR, Hyp Auxiliary Control Register* on page B1-196
HACTLR2	RW	*B1.47 HACTLR2, Hyp Auxiliary Control Register 2* on page B1-198
HADFSR	RW	*B1.48 HADFSR, Hyp Auxiliary Data Fault Status Syndrome Register* on page B1-199
HAISFR	RW	*B1.49 HAIFSR, Hyp Auxiliary Instruction Fault Status Syndrome Register* on page B1-200
HAMAIR0	RW	*B1.50 HAMAIR0, Hyp Auxiliary Memory Attribute Indirection Register 0* on page B1-201
HAMAIR1	RW	*B1.51 HAMAIR1, Hyp Auxiliary Memory Attribute Indirection Register 1* on page B1-202
HCR	RW	*B1.52 HCR, Hyp Configuration Register* on page B1-203
HCR2	RW	*B1.53 HCR2, Hyp Configuration Register 2* on page B1-205
HSR	RW	*B1.55 HSR, Hyp Syndrome Register* on page B1-208
HTTBR	RW	*B1.56 HTTBR, Hyp Translation Table Base Register* on page B1-210
VDFSR	RW	*B1.88 VDFSR, Virtual SError Exception Syndrome Register* on page B1-263
VDISR	RW	*B1.89 VDISR, Virtual Deferred Interrupt Status Register* on page B1-264
VMPIDR	RW	*B1.90 VMPIDR, Virtualization Multiprocessor ID Register* on page B1-267
VPIDR	RW	*B1.91 VPIDR, Virtualization Processor ID Register* on page B1-268
VTCR	RW	*B1.92 VTCR, Virtualization Translation Control Register* on page B1-269
VTTBR	RW	*B1.93 VTTBR, Virtualization Translation Table Base Register* on page B1-272

Exception and fault handling registers

Name	Type	Description
ADFSR	RW	*B1.7 ADFSR, Auxiliary Data Fault Status Register* on page B1-139
AIFSR	RW	*B1.9 AIFSR, Auxiliary Instruction Fault Status Register* on page B1-141

(continued)

Name	Type	Description
DFSR	RW	*B1.26 DFSR, Data Fault Status Register* on page B1-171
DISR	RW	*B1.27 DISR, Deferred Interrupt Status Register* on page B1-173
HADFSR	RW	*B1.48 HADFSR, Hyp Auxiliary Data Fault Status Syndrome Register* on page B1-199
HAIFSR	RW	*B1.49 HAIFSR, Hyp Auxiliary Instruction Fault Status Syndrome Register* on page B1-200
HSR	RW	*B1.55 HSR, Hyp Syndrome Register* on page B1-208
IFSR	RW	*B1.73 IFSR, Instruction Fault Status Register* on page B1-240
VDFSR	RW	*B1.88 VDFSR, Virtual SError Exception Syndrome Register* on page B1-263
VDISR	RW	*B1.89 VDISR, Virtual Deferred Interrupt Status Register* on page B1-264

Implementation defined registers

Name	Type	Description
CPUACTLR	RW	*B1.15 CPUACTLR, CPU Auxiliary Control Register* on page B1-149
CPUACTLR2	RW	*B1.16 CPUACTLR2, CPU Auxiliary Control Register 2* on page B1-151
CPUCFR	RO	*B1.17 CPUCFR, CPU Configuration Register* on page B1-153
ERXPFGCDNR	RW	*B1.40 ERXPFGCDNR, Selected Error Pseudo Fault Generation Count Down Register* on page B1-188
ERXPFGCTLR	RW	*B1.41 ERXPFGCTLR, Selected Error Pseudo Fault Generation Control Register* on page B1-190
ERXPFGFR	RO	*B1.42 ERXPFGFR, Selected Pseudo Fault Generation Feature Register* on page B1-192
CPUECTLR	RW	*B1.18 CPUECTLR, CPU Extended Control Register* on page B1-155
CPUPCR	RW	*B1.19 CPUPCR, CPU Private Control Register* on page B1-158
CPUPMR	RW	*B1.20 CPUPMR, CPU Private Mask Register* on page B1-160
CPUPOR	RW	*B1.21 CPUPOR, CPU Private Operation Register* on page B1-162
CPUPSELR	RW	*B1.22 CPUPSELR, CPU Private Selection Register* on page B1-164

The following table shows the 32-bit wide implementation defined Cluster registers. These registers are RW, and details can be found in *Arm® DynamIQ™ Shared Unit Technical Reference Manual*

Table B1-5 Cluster registers

Name	Copro	CRn	Opc1	CRm	Opc2	Width	Description
CLUSTERCFR	cp15	c15	0	c3	0	32-bit	Cluster configuration register.
CLUSTERIDR	cp15	c15	0	c3	1	32-bit	Cluster main revision ID.
CLUSTEREVIDR	cp15	c15	0	c3	2	32-bit	Cluster ECO ID.
CLUSTERACTLR	cp15	c15	0	c3	3	32-bit	Cluster auxiliary control register.
CLUSTERECTLR	cp15	c15	0	c3	4	32-bit	Cluster extended control register.
CLUSTERPWRCTLR	cp15	c15	0	c3	5	32-bit	Cluster power control register.
CLUSTERPWRDN	cp15	c15	0	c3	6	32-bit	Cluster power down register.
CLUSTERPWRSTAT	cp15	c15	0	c3	7	32-bit	Cluster power status register.

Table B1-5 Cluster registers (continued)

Name	Copro	CRn	Opc1	CRm	Opc2	Width	Description
CLUSTERTHREADSID	cp15	c15	0	c4	0	32-bit	Cluster thread scheme ID register.
CLUSTERACPSID	cp15	c15	0	c4	1	32-bit	Cluster ACP scheme ID register.
CLUSTERSTASHSID	cp15	c15	0	c4	2	32-bit	Cluster stash scheme ID register.
CLUSTERPARTCR	cp15	c15	0	c4	3	32-bit	Cluster partition control register.
CLUSTERBUSQOS	cp15	c15	0	c4	4	32-bit	Cluster bus QoS control register.
CLUSTERL3HIT	cp15	c15	0	c4	5	32-bit	Cluster L3 hit counter register.
CLUSTERL3MISS	cp15	c15	0	c4	6	32-bit	Cluster L3 miss counter register.
CLUSTERTHREADSIDOVR	cp15	c15	0	c4	7	32-bit	Cluster thread scheme ID override register.
CLUSTERPM*	cp15	c15	0 or 6	c5-c6	0-7	32-bit or 64-bit	Cluster PMU registers

Legacy feature registers

Name	Type	Description
FCSEIDR	RO	*B1.44 FCSEIDR, FCSE Process ID Register* on page B1-194 In Armv8, the core does not implement the FCSEIDR, and therefore the register is RO.

Address registers

Name	Type	Description
PAR	RW	*B1.78 PAR, Physical Address Register* on page B1-247

B1.5 ACTLR, Auxiliary Control Register

The ACTLR provides access control for IMPLEMENTATION DEFINED registers at lower exception levels.

ACTLR is a 32-bit register, and is part of:

- The Other system control registers functional group.
- The Implementation defined functional group.

Bit field descriptions

The core implements the ACTLR(NS) register, but has no defined bits. This register is always RES0.

The following bit field descriptions are for the Secure version of the ACTLR.

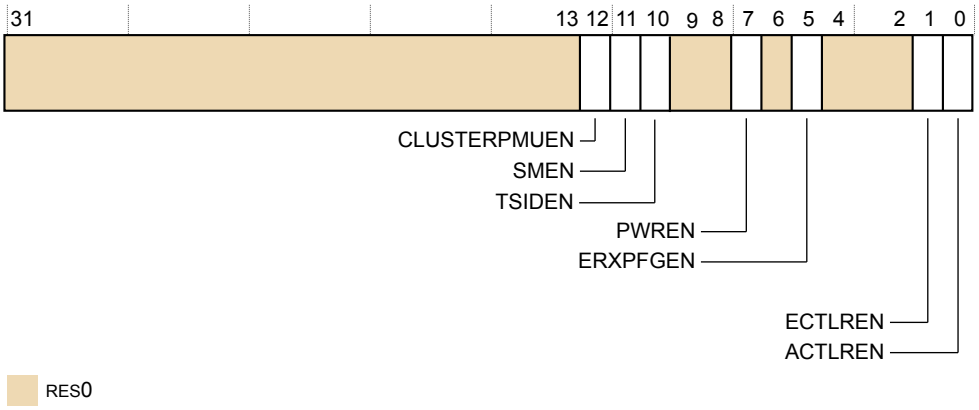

Figure B1-1 ACTLR (S) bit assignments

RES0, [31:13]

RES0 Reserved.

CLUSTERPMUEN, [12]

Performance Management Registers enable. The value is:

0 CLUSTERPM* registers are not write accessible from a lower Exception level. This is the reset value.

1 CLUSTERPM* registers are write accessible from EL2.

SMEN, [11]

Scheme Management Registers enable. The value is:

0 Registers CLUSTERTHREADSID, CLUSTERACPSID, CLUSTERSTASHSID, CLUSTERPARTCR, and CLUSTERBUSQOS are not write accessible from EL2. This is the reset value.

1 Registers controlled by the TSIDEN bit, CLUSTERACPSID, CLUSTERSTASHSID, CLUSTERPARTCR, and CLUSTERBUSQOS are write accessible from EL2.

TSIDEN, [10]

Thread Scheme ID Register enable. The possible values are:

0 Register CLUSTERTHREADSID is not accessible from EL1 nonsecure. This is the reset value.

1 Register CLUSTERTHREADSID is accessible from EL1 nonsecure if they are write accessible from EL2.

RES0, [9:8]

RES0 Reserved.

PWREN, [7]

Power Control Registers enable. The value is:

0 Registers CPUPWRCTLR, CLUSTERPWRCTLR, CLUSTERPWRDN, CLUSTERPWRSTAT, CLUSTERL3HIT and CLUSTERL3MISS are not write accessible from a lower Exception level. This is the reset value.

1 Registers CPUPWRCTLR, CLUSTERPWRCTLR, CLUSTERPWRDN, CLUSTERPWRSTAT, CLUSTERL3HIT and CLUSTERL3MISS are write accessible from EL2.

RES0, [6]

RES0 Reserved.

ERXPFGEN, [5]

Error Record Registers enable. The value is:

0 ERXPFG* are not write accessible from a lower Exception level. This is the reset value.

1 ERXPFG* are write accessible from EL2.

RES0, [4:2]

RES0 Reserved.

ECTLREN, [1]

Extended Control Registers enable. The value is:

0 CPUECTLR and CLUSTERECTLR are not write accessible from a lower Exception level. This is the reset value.

1 CPUECTLR and CLUSTERECTLR are write accessible from EL2.

ACTLREN, [0]

Auxiliary Control Registers enable. The value is:

0 CPUACTLR, CPUACTLR2 and CLUSTERACTLR are not write accessible from a lower Exception level. This is the reset value.

1 CPUACTLR, CPUACTLR2 and CLUSTERACTLR are write accessible from EL2.

Configurations

AArch32 register ACTLR(NS) is mapped to AArch64 register ACTLR_EL1. See *B2.5 ACTLR_EL1, Auxiliary Control Register, EL1* on page B2-293.

AArch32 register ACTLR(S) is mapped to AArch64 register ACTLR_EL3. See *B2.7 ACTLR_EL3, Auxiliary Control Register, EL3* on page B2-296.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

B1.6 ACTLR2, Auxiliary Control Register 2

The core does not use ACTLR2. This register is always RES0.

B1.7 ADFSR, Auxiliary Data Fault Status Register

The ADFSR provides extra IMPLEMENTATION DEFINED fault status information for Data Abort exceptions taken to EL1 modes. In the Cortex-A75 core, no additional information is provided for such exceptions, so this register is not used.

Bit field descriptions

ADFSR is a 32-bit register, and is part of:

- The Exception and fault handling registers functional group.
- The Implementation defined functional group.

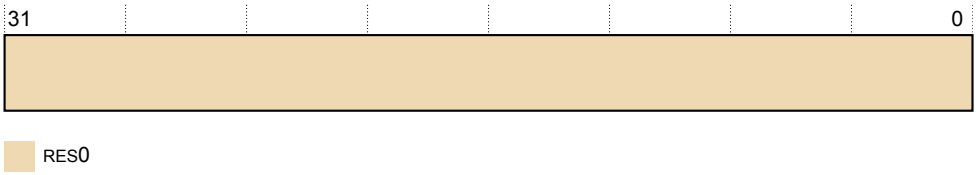

Figure B1-2 ADFSR bit assignments

[31:0]

Reserved, RES0.

Configurations

AArch32 System register ADFSR is architecturally mapped to AArch64 System register AFSR0_EL1. See *B2.8 AFSR0_EL1, Auxiliary Fault Status Register 0, EL1* on page B2-298.

AArch32 register ADFSR(S) is architecturally mapped to AArch64 register AFSR0_EL3. See *B2.10 AFSR0_EL3, Auxiliary Fault Status Register 0, EL3* on page B2-300.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

B1.8 AIDR, Auxiliary ID Register

The AIDR provides IMPLEMENTATION DEFINED identification information. This register is not used in the A75 core.

Bit field descriptions

AIDR is a 32-bit register, and is part of:

- The Identification registers functional group.
- The Implementation defined functional group.

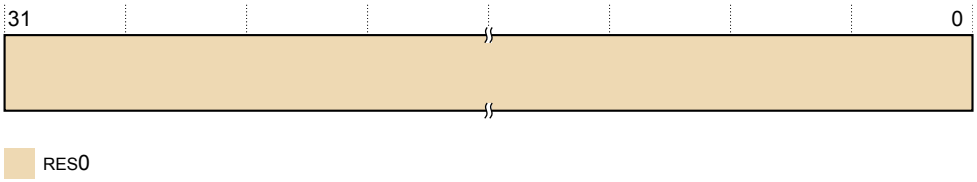

Figure B1-3 AIDR bit assignments

RES0, [31:0]

Reserved, RES0.

Configurations

There is one instance of this register that is used in both Secure and Non-secure states.

AArch32 System register AIDR is architecturally mapped to AArch64 System register AIDR_EL1. See *B2.14 AIDR_EL1, Auxiliary ID Register, EL1* on page B2-304.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

B1.9 AIFSR, Auxiliary Instruction Fault Status Register

The AIFSR provides extra IMPLEMENTATION DEFINED fault status information for Prefetch Abort exceptions that are taken to EL1 modes. This register is not used in the Cortex-A75 core.

Bit field descriptions

AIFSR is a 32-bit register, and is part of:

- The Exception and fault handling registers functional group.
- The Implementation defined functional group.

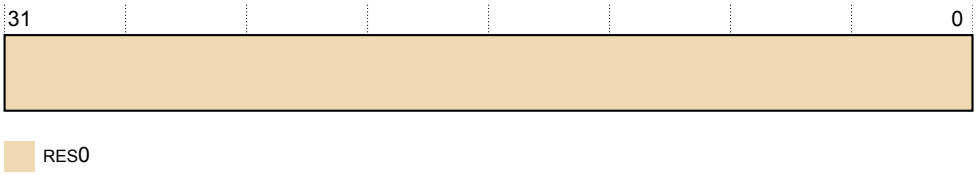

Figure B1-4 AIFSR bit assignments

RES0, [31:0]

RES0 Reserved.

Configurations

AArch32 System register AIFSR is architecturally mapped to AArch64 System register AFSR1_EL1. See *B2.11 AFSR1_EL1, Auxiliary Fault Status Register 1, EL1* on page B2-301.

AArch32 System register AIFSR(S) is architecturally mapped to AArch64 System register AFSR1_EL3. See *B2.13 AFSR1_EL3, Auxiliary Fault Status Register 1, EL3* on page B2-303.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

B1.10 AMAIR0, Auxiliary Memory Attribute Indirection Register 0

When using the Long-descriptor format translation tables for stage 1 translations, AMAIR0 provides IMPLEMENTATION DEFINED memory attributes for the memory regions specified by MAIR0. This register is not used in the Cortex-A75 core.

Bit field descriptions

AMAIR0 is a 32-bit register, and is part of:

- The Virtual memory control registers functional group.
- The Implementation defined functional group.

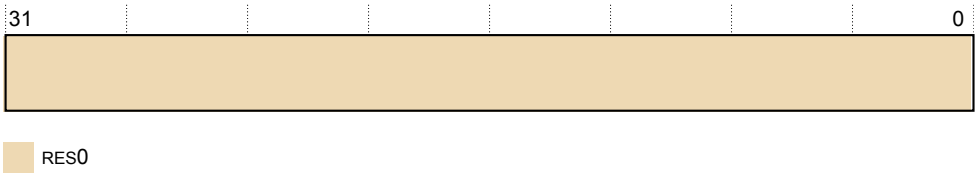

Figure B1-5 AMAIR0 bit assignments

RES0, [31:0]

RES0 Reserved.

Configurations

AArch32 System register AMAIR0 is architecturally mapped to AArch64 System register AMAIR_EL1[31:0]. See *B2.15 AMAIR_EL1, Auxiliary Memory Attribute Indirection Register, EL1* on page B2-305.

AArch32 System register AMAIR0(S) is architecturally mapped to AArch64 System register AMAIR_EL3[31:0]. See *B2.17 AMAIR_EL3, Auxiliary Memory Attribute Indirection Register, EL3* on page B2-307.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

B1.11 AMAIR1, Auxiliary Memory Attribute Indirection Register 1

When using the Long-descriptor format translation tables for stage 1 translations, AMAIR1 provides IMPLEMENTATION DEFINED memory attributes for the memory regions specified by MAIR1. This register is not used in the Cortex-A75 core.

Bit field descriptions

AMAIR1 is a 32-bit register, and is part of:

- The Virtual memory control registers functional group.
- The Implementation defined functional group.

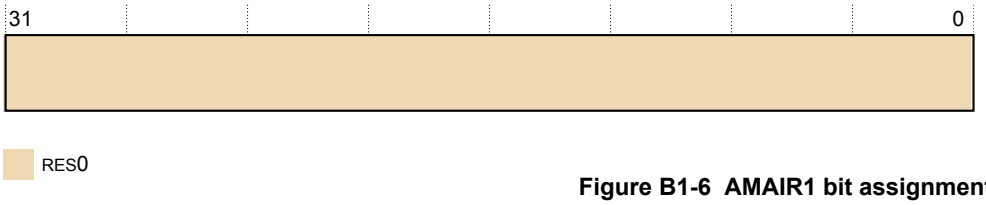

Figure B1-6 AMAIR1 bit assignments

RES0, [31:0]

RES0 Reserved.

Configurations

AArch32 System register AMAIR1 is architecturally mapped to AArch64 System register AMAIR_EL1[63:32]. See *B2.15 AMAIR_EL1, Auxiliary Memory Attribute Indirection Register, EL1* on page B2-305.

AArch32 System register AMAIR1(S) is architecturally mapped to AArch64 System register AMAIR_EL3[63:32]. See *B2.17 AMAIR_EL3, Auxiliary Memory Attribute Indirection Register, EL3* on page B2-307.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

B1.12 CCSIDR, Cache Size ID Register

The CCSIDR provides information about the architecture of the currently selected cache.

Bit field descriptions

CCSIDR is a 32-bit register and is part of the Identification registers functional group.

This register is Read Only.

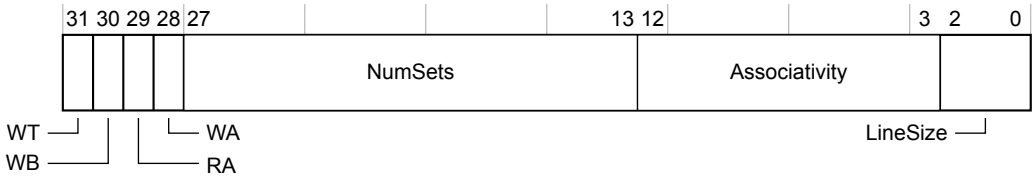

Figure B1-7 CCSIDR bit assignments

WT, [31]
Indicates whether the selected cache level supports Write-Through:

`0` Cache Write-Through is not supported at any level.

For more information about encoding, see *Table B1-6 CCSIDR encodings* on page B1-145.

WB, [30]
Indicates whether the selected cache level supports Write-Back. Permitted values are:

`0` Write-Back is not supported.

`1` Write-Back is supported.

For more information about encoding, see *Table B1-6 CCSIDR encodings* on page B1-145.

RA, [29]
Indicates whether the selected cache level supports read-allocation. Permitted values are:

`0` Read-allocation is not supported.

`1` Read-allocation is supported.

For more information about encoding, see *Table B1-6 CCSIDR encodings* on page B1-145.

WA, [28]
Indicates whether the selected cache level supports write-allocation. Permitted values are:

`0` Write-allocation is not supported.

`1` Write-allocation is supported.

For more information about encoding, see *Table B1-6 CCSIDR encodings* on page B1-145.

NumSets, [27:13]
(Number of sets in cache) - 1. Therefore, a value of 0 indicates one set in the cache. The number of sets does not have to be a power of 2.

For more information about encoding, see *Table B1-6 CCSIDR encodings* on page B1-145.

Associativity, [12:3]

(Associativity of cache) - 1. Therefore, a value of 0 indicates an associativity of 1. The associativity does not have to be a power of 2.

For more information about encoding, see *Table B1-6 CCSIDR encodings* on page B1-145.

LineSize, [2:0]

(Log_2(Number of bytes in cache line)) - 4. For example:

Indicates the (log_2 (number of words in cache line)) - 2:

For a line length of 16 bytes: $Log_2(16) = 4$, LineSize entry = 0. This is the minimum line length.

For a line length of 32 bytes: $Log_2(32) = 5$, LineSize entry = 1.

For more information about encoding, see *Table B1-6 CCSIDR encodings* on page B1-145.

Configurations

CCSIDR is architecturally mapped to AArch64 register CCSIDR_EL1. See *B2.18 CCSIDR_EL1, Cache Size ID Register, EL1* on page B2-308.

There is one copy of this register that is used in both Secure and Non-secure states.

The implementation includes one CCSIDR for each cache that it can access. CSSELR selects which Cache Size ID Register is accessible.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

CCSIDR encodings

The following table shows the individual bit field and complete register encodings for the CCSIDR. The CSSELR determines which CCSIDR to select.

Table B1-6 CCSIDR encodings

CSSELR		Cache	Size	Complete register encoding	Register bit field encoding						
Level	InD				WT	WB	RA	WA	NumSets	Associativity	LineSize
0b000	0b0	L1 Data cache	64KB	7007E07A	0	1	1	1	0x003F	0x00F	2
0b000	0b1	L1 Instruction cache	64KB	201FE01A	0	0	1	0	0x00FF	0x003	2
0b001	0b0	L2 cache	256KB	703FE03A	0	1	1	1	0x01FF	0x007	2
			512KB	707FE03A	0	1	1	1	0x03FF	0x007	2
0b001	0b1	Reserved	-	-	-	-	-	-	-	-	-
0b010	0b0	L3 cache When L3 is not implemented, the value of CCSIDR is UNKNOWN.	512KB	703FE07A	0	1	1	1	0x01FF	0x00F	2
			1024KB	707FE07A					0x03FF		
			2048KB	70FFE07A					0x07FF		
			4096KB	71FFE07A					0x0FFF		
0b010	0b1	Reserved	-	-	-	-	-	-	-	-	-
0b0101 - 0b1111		Reserved	-	-	-	-	-	-	-	-	-

B1.13 CLIDR, Cache Level ID Register

The CLIDR identifies the type of cache, or caches, implemented at each level, up to a maximum of seven levels.

It also identifies the *Level of Coherency* (LoC) and *Level of Unification* (LoU) for the cache hierarchy.

Bit field descriptions

CLIDR is a 32-bit register, and is part of the Identification registers functional group.

This register is Read Only.

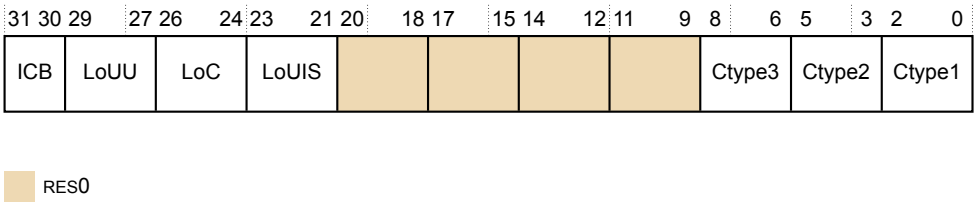

Figure B1-8 CLIDR bit assignments

ICB, [31:30]
Inner cache boundary. This field indicates the boundary between the inner and the outer domain:

`0b10` L2 cache is the highest inner level.

`0b11` L3 cache is the highest inner level.

LoUU, [29:27]
Indicates the Level of Unification Uniprocessor for the cache hierarchy:

`0b000` L1 is the level of Unification Uniprocessor.

LoC, [26:24]
Indicates the Level of Coherency for the cache hierarchy:

`0b010` L3 cache is not implemented.

`0b011` L3 cache is implemented.

LoUIS, [23:21]
Indicates the *Level of Unification Inner Shareable* (LoUIS) for the cache hierarchy.

`0b000` No cache level needs cleaning to Point of Unification.

RES0, [20:9]
No cache at levels L7 down to L4.

RES0 Reserved.

Ctype3, [8:6]
Indicates the type of cache if the core implements L3 cache. If present, unified instruction and data caches at level 3:

`0b000` L3 cache is not implemented.

`0b100` L3 cache is implemented.

Ctype2, [5:3]
Indicates the type of unified instruction and data caches at level 2:

`0b100` L2 cache is implemented as a unified cache.

Ctype1, [2:0]

Indicates the type of cache implemented at L1:

`0b011` Separate instruction and data caches at L1.

Configurations

CLIDR is architecturally mapped to AArch64 register CLIDR_EL1. See *B2.19 CLIDR_EL1, Cache Level ID Register, EL1* on page B2-310.

There is one copy of this register that is used in both Secure and Non-secure states.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

B1.14 CPACR, Architectural Feature Access Control Register

The CPACR controls access to floating-point, and Advanced SIMD functionality from EL0, EL1, and EL3.

Bit field descriptions

CPACR is a 32-bit register, and is part of the Other system control registers functional group.

This register resets to value `0x00000000`.

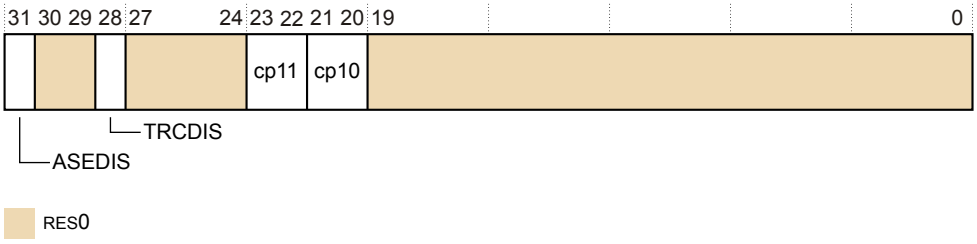

Figure B1-9 CPACR bit assignments

TRCDIS, [28]

This bit is reserved, RES0.

Configurations

CPACR is architecturally mapped to AArch64 register CPACR_EL1. See *B2.20 CPACR_EL1, Architectural Feature Access Control Register, EL1* on page B2-312.

There is one copy of this register that is used in both Secure and Non-secure states.

Bits in the NSACR control Non-secure access to the CPACR fields. See the field descriptions cp10 and cp11.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

B1.15 CPUACTLR, CPU Auxiliary Control Register

The CPUACTLR provides IMPLEMENTATION DEFINED configuration and control options for the core.

Bit field descriptions

CPUACTLR is a 64-bit register, and is part of the Implementation defined registers functional group.

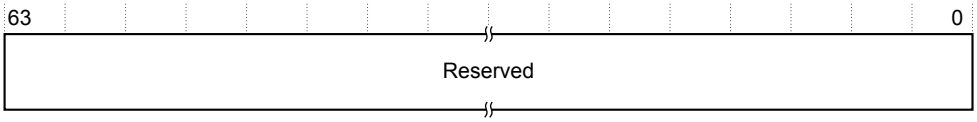

Figure B1-10 CPUACTLR bit assignments

Reserved, [63:0]

Reserved for Arm internal use.

Configurations

CPUACTLR is:

- Common to the Secure and Non-secure states.
- Mapped to the AArch64 CPUACTLR_EL1 register. See *B2.23 CPUACTLR_EL1, CPU Auxiliary Control Register, EL1* on page B2-315.

Usage constraints

Accessing the CPUACTLR

Arm recommends that you write to this register after a powerup reset, before the MMU is enabled.

Setting many of these bits can cause significantly lower performance on your code. Therefore, Arm strongly recommends that you do not modify this register unless directed by Arm.

This register can be read using MRRC with the following syntax:

```
MRRC <syntax>
```

This register can be written using MCRR with the following syntax:

```
MCRR <syntax>
```

This syntax is encoded with the following settings in the instruction encoding:

<syntax>	coproc	opc1	CRm
p15, 0, <Rt>, <Rt2>, c15	1111	0000	1111

Accessibility

This register is accessible in software as follows:

<syntax>	Control			Accessibility			
	E2H	TGE	NS	EL0	EL1	EL2	EL3
p15, 0, <Rt>, <Rt2>, c15	x	x	0	-	RW	n/a	RW
p15, 0, <Rt>, <Rt2>, c15	x	0	1	-	RW	RW	RW
p15, 0, <Rt>, <Rt2>, c15	x	1	1	-	n/a	RW	RW

'n/a' Not accessible. The core cannot be executing at this Exception level, so this access is not possible.

Traps and enables

For a description of the prioritization of any generated exceptions, see *Exception priority order* in the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile* for exceptions taken to AArch32 state, and see *Synchronous exception prioritization* for exceptions taken to AArch64 state.

Write access to this register from EL1 or EL2 depends on the value of bit[0] of ACTLR_EL2, ACTLR_EL3, ACTLR (S), and HACTLR.

B1.16 CPUACTLR2, CPU Auxiliary Control Register 2

The CPUACTLR2 provides IMPLEMENTATION DEFINED configuration and control options for the core.

Bit field descriptions

CPUACTLR2 is a 64-bit register, and is part of the Implementation defined registers functional group.

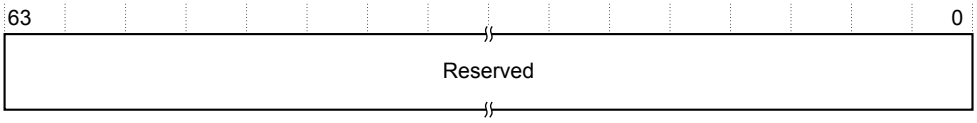

Figure B1-11 CPUACTLR2 bit assignments

Reserved, [63:0]

Reserved for Arm internal use.

Configurations

CPUACTLR2 is:

- Common to the Secure and Non-secure states.
- Mapped to the AArch64 CPUACTLR2_EL1 register. See *B2.24 CPUACTLR2_EL1, CPU Auxiliary Control Register 2, EL1* on page B2-317.

Usage constraints

Accessing the CPUACTLR2

Arm recommends that you write to this register after a powerup reset, before the MMU is enabled.

Setting many of these bits can cause significantly lower performance on your code. Therefore, Arm strongly recommends that you do not modify this register unless directed by Arm.

This register can be read using MRRC with the following syntax:

```
MRRC <syntax>
```

This register can be written using MCRR with the following syntax:

```
MCRR <syntax>
```

This syntax is encoded with the following settings in the instruction encoding:

<syntax>	coproc	opc1	CRm
p15, 1, <Rt>, <Rt2>, c15	1111	0001	1111

Accessibility

This register is accessible in software as follows:

<syntax>	Control			Accessibility			
	E2H	TGE	NS	EL0	EL1	EL2	EL3
p15, 1, <Rt>, <Rt2>, c15	x	x	0	-	RW	n/a	RW
p15, 1, <Rt>, <Rt2>, c15	x	0	1	-	RW	RW	RW
p15, 1, <Rt>, <Rt2>, c15	x	1	1	-	n/a	RW	RW

'n/a' Not accessible. The core cannot be executing at this Exception level, so this access is not possible.

Traps and enables

For a description of the prioritization of any generated exceptions, see *Exception priority order* in the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile* for exceptions taken to AArch32 state, and see *Synchronous exception prioritization* for exceptions taken to AArch64 state.

Write access to this register from EL1 or EL2 depends on the value of bit[0] of ACTLR_EL2, ACTLR_EL3, ACTLR (S), and HACTLR.

B1.17 CPUCFR, CPU Configuration Register

The CPUCFR provides configuration information for the core.

Bit field descriptions

CPUCFR is a 32-bit register and is part of the Implementation registers functional group.

This register is Read Only.

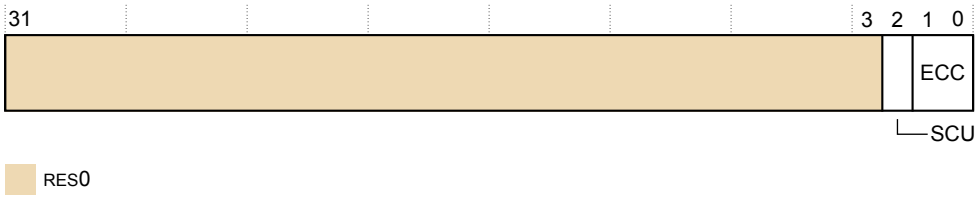

Figure B1-12 CPUCFR bit assignments

RES0, [31:3]

RES0 Reserved.

SCU, [2]

Indicates whether the SCU is present or not. The value is:

0 The SCU is present.

ECC, [1:0]

Indicates whether ECC is present or not. The possible values are:

0 ECC is not present.

1 ECC is present.

Configurations

CPUCFR is architecturally mapped to AArch64 register CPUCFR_EL1. See *B2.25 CPUCFR_EL1, CPU Configuration Register, EL1* on page B2-319.

Usage constraints

Accessing the CPUCFR

This register can be read using MRC with the following syntax:

```
MRC <syntax>
```

This syntax is encoded with the following settings in the instruction encoding:

<syntax>	coproc	opc1	CRn	CRm	opc2
p15, 0, <Rt>, c15, c0, 0	1111	000	1111	0000	000

Accessibility

This register is accessible in software as follows:

<syntax>	Control			Accessibility			
	E2H	TGE	NS	EL0	EL1	EL2	EL3
p15, 0, <Rt>, c15, c0, 0	x	x	0	-	RO	n/a	RO
p15, 0, <Rt>, c15, c0, 0	x	0	1	-	RO	RO	RO
p15, 0, <Rt>, c15, c0, 0	x	1	1	-	n/a	RO	RO

'n/a' Not accessible. The PE cannot be executing at this Exception level, so this access is not possible.

B1.18 CPUECTLR, CPU Extended Control Register

The CPUECTLR provides additional IMPLEMENTATION DEFINED configuration and control options for the core.

Bit field descriptions

CPUECTLR is a 64-bit register, and is part of the 64-bit registers functional group.

This register resets to value `0x0000FFF0`.

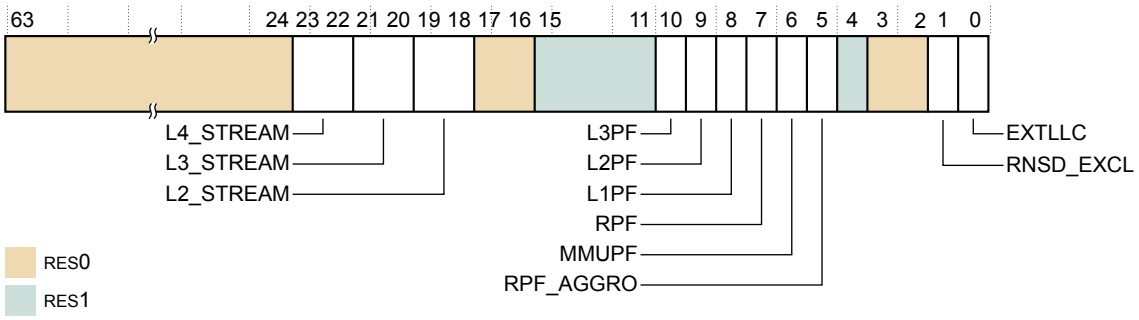

Figure B1-13 CPUECTLR bit assignments

RES0, [63:24]

RES0 Reserved.

L4_STREAM, [23:22]

Threshold for direct stream to L4 cache on store. The possible values are:

`0b00` 512KB.

`0b01` 1024KB.

`0b10` 2048KB.

`0b11` Stream disabled.

L3_STREAM, [21:20]

Threshold for direct stream to L3 cache on store. The possible values are:

`0b00` 64KB.

`0b01` 256KB.

`0b10` 512KB.

`0b11` Stream disabled.

L2_STREAM, [19:18]

Threshold for direct stream to L2 cache on store. The possible values are:

`0b00` 16KB.

`0b01` 64KB.

`0b10` 128KB.

`0b11` Stream disabled.

RES0, [17:16]

RES0 Reserved.

RES1, [15:11]

RES1 Reserved.

L3PF, [10]

Enable L3 prefetch requests sent by the stride prefetcher. The possible values are:

0 L3 prefetch requests are disabled.

1 L3 prefetch requests are enabled. This is the reset value.

L2PF, [9]

Enable L2 prefetch requests sent by the stride prefetcher. The possible values are:

0 L2 prefetch requests are disabled.

1 L2 prefetch requests are enabled. This is the reset value.

L1PF, [8]

Enable L1 prefetch requests sent by the stride prefetcher. The possible values are:

0 L1 prefetch requests are disabled.

1 L1 prefetch requests are enabled. This is the reset value.

RPF, [7]

Enable L2 region prefetch requests. The possible values are:

0 L2 region prefetch requests are disabled.

1 L2 region prefetch requests are enabled. This is the reset value.

MMUPF, [6]

Enable MMU prefetch requests. The possible values are:

0 MMU prefetch requests are disabled.

1 MMU prefetch requests are enabled. This is the reset value.

RPF_AGGRO, [5]

L2 region prefetcher aggressivity. The possible values are:

0 The L2 region prefetcher is less aggressive, with a longer learning phase.

1 The L2 region prefetcher is more aggressive, with a shorter learning phase. This is the reset value.

RES1, [4]

RES1 Reserved.

RES0, [3:2]

RES0 Reserved.

RNSD_EXCL, [1]

Enables signaling of cacheable Exclusive loads on the internal interface between the core and the DSU. The possible values are:

`0x0` Cacheable Exclusive Loads do not use the Exclusive attribute on the internal interface between the core and the DSU. This is the reset value.

`0x1` Cacheable Exclusive Loads use the Exclusive attribute on the internal interface between the core and the DSU.

EXTLLC, [0]

The possible values are:

`0x0` Indicates that an internal Last-level cache is present in the system, and that the DataSource field on the master CHI interface indicates when data is returned from the LLC. This is used to control how the LL_CACHE* PMU events count. This is the reset value.

`0x1` Indicates that an external Last-level cache is present in the system, and that the DataSource field on the master CHI interface indicates when data is returned from the LLC. This is used to control how the LL_CACHE* PMU events count.

Configurations

The CPUECTLR is mapped to the AArch64 CPUECTLR_EL1 register. See *B2.26 CPUECTLR_EL1, CPU Extended Control Register, EL1* on page B2-321.

Usage constraints

Accessing the CPUECTLR

The CPUECTLR can be written dynamically.

This register can be read using MRRC with the following syntax:

```
MRRC <syntax>
```

This register can be written using MCRR with the following syntax:

```
MCRR <syntax>
```

This syntax is encoded with the following settings in the instruction encoding:

<syntax>	coproc	opc1	CRm
p15, 4, <Rt>, <Rt2>, c15	1111	0100	1111

Accessibility

This register is accessible in software as follows:

<syntax>	Control			Accessibility			
	E2H	TGE	NS	EL0	EL1	EL2	EL3
p15, 4, <Rt>, <Rt2>, c15	x	x	0	-	RW	n/a	RW
p15, 4, <Rt>, <Rt2>, c15	x	0	1	-	RW	RW	RW
p15, 4, <Rt>, <Rt2>, c15	x	1	1	-	n/a	RW	RW

'n/a' Not accessible. The PE cannot be executing at this Exception level, so this access is not possible.

Traps and enables

For a description of the prioritization of any generated exceptions, see *Exception priority order* in the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile* for exceptions taken to AArch32 state, and see *Synchronous exception prioritization* for exceptions taken to AArch64 state.

Access to this register depends on bit[1] of ACTLR_EL2, ACTLR_EL3, ACTLR (S), and HACTLR.

B1.19 CPUPCR, CPU Private Control Register

The CPUPCR provides IMPLEMENTATION DEFINED configuration and control options for the core.

Bit field descriptions

CPUPCR is a 64-bit register, and is part of the Implementation defined registers functional group.

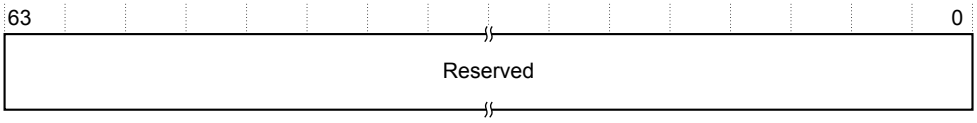

Figure B1-14 CPUPCR bit assignments

Reserved, [63:0]

Reserved for Arm internal use.

Configurations

CPUPCR is:

- Only accessible in Secure state.
- Mapped to the AArch64 CPUPCR_EL3 register. See *B2.27 CPUPCR_EL3, CPU Private Control Register, EL3* on page B2-324.

Usage constraints

Accessing the CPUPCR

Arm recommends that you write to this register after a powerup reset, before the MMU is enabled.

Writing to this register might cause UNPREDICTABLE behaviors. Therefore, Arm strongly recommends that you do not modify this register unless directed by Arm.

This register can be read using MRRC with the following syntax:

```
MRRC <syntax>
```

This register can be written using MCRR with the following syntax:

```
MCRR <syntax>
```

This syntax is encoded with the following settings in the instruction encoding:

<syntax>	coproc	opc1	CRm
p15, 8, <Rt>, <Rt2>, c15	1111	1000	1111

Accessibility

This register is accessible in software as follows:

<syntax>	Control			Accessibility			
	E2H	TGE	NS	EL0	EL1	EL2	EL3
p15, 8, <Rt>, <Rt2>, c15	x	x	0	-	-	n/a	RW
p15, 8, <Rt>, <Rt2>, c15	x	0	1	-	-	-	RW
p15, 8, <Rt>, <Rt2>, c15	x	1	1	-	n/a	-	RW

'n/a' Not accessible. The core cannot be executing at this Exception level, so this access is not possible.

Traps and enables

For a description of the prioritization of any generated exceptions, see *Exception priority order* in the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile* for exceptions taken to AArch32 state, and see *Synchronous exception prioritization* for exceptions taken to AArch64 state.

B1.20 CPUPMR, CPU Private Mask Register

The CPUPMR provides IMPLEMENTATION DEFINED configuration and control options for the core.

Bit field descriptions

CPUPMR is a 64-bit register, and is part of the Implementation defined registers functional group.

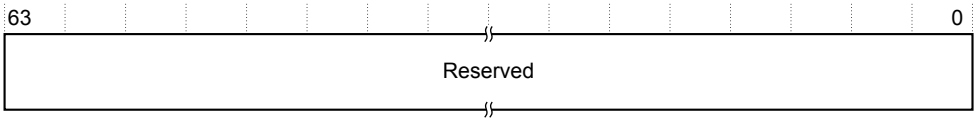

Figure B1-15 CPUPMR bit assignments

Reserved, [63:0]

Reserved for Arm internal use.

Configurations

CPUPMR is:

- Only accessible in Secure state.
- Mapped to the AArch64 CPUPMR_EL3 register. See *B2.28 CPUPMR_EL3, CPU Private Mask Register, EL3* on page B2-326.

Usage constraints

Accessing the CPUPOR

Arm recommends that you write to this register after a powerup reset, before the MMU is enabled.

Writing to this register might cause UNPREDICTABLE behaviors. Therefore, Arm strongly recommends that you do not modify this register unless directed by Arm.

This register can be read using MRRC with the following syntax:

```
MRRC <syntax>
```

This register can be written using MCRR with the following syntax:

```
MCRR <syntax>
```

This syntax is encoded with the following settings in the instruction encoding:

<syntax>	coproc	opc1	CRm
p15, 10, <Rt>, <Rt2>, c15	1111	1010	1111

Accessibility

This register is accessible in software as follows:

<syntax>	Control			Accessibility			
	E2H	TGE	NS	EL0	EL1	EL2	EL3
p15, 10, <Rt>, <Rt2>, c15	x	x	0	-	-	n/a	RW
p15, 10, <Rt>, <Rt2>, c15	x	0	1	-	-	-	RW
p15, 10, <Rt>, <Rt2>, c15	x	1	1	-	n/a	-	RW

'n/a' Not accessible. The core cannot be executing at this Exception level, so this access is not possible.

Traps and enables

For a description of the prioritization of any generated exceptions, see *Exception priority order* in the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile* for exceptions taken to AArch32 state, and see *Synchronous exception prioritization* for exceptions taken to AArch64 state.

B1.21 CPUPOR, CPU Private Operation Register

The CPUPOR provides IMPLEMENTATION DEFINED configuration and control options for the core.

Bit field descriptions

CPUPOR is a 64-bit register, and is part of the Implementation defined registers functional group.

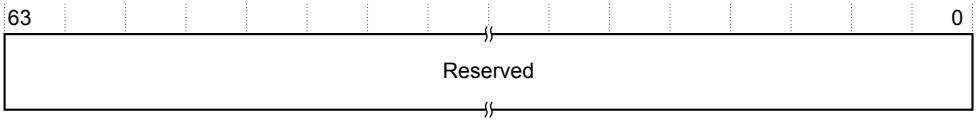

Figure B1-16 CPUPOR bit assignments

Reserved, [63:0]

Reserved for Arm internal use.

Configurations

CPUPOR is:

- Only accessible in Secure state.
- Mapped to the AArch64 CPUPOR_EL3 register. See *B2.29 CPUPOR_EL3, CPU Private Operation Register, EL3* on page B2-328.

Usage constraints

Accessing the CPUPOR

Arm recommends that you write to this register after a powerup reset, before the MMU is enabled.

Writing to this register might cause UNPREDICTABLE behaviors. Therefore, Arm strongly recommends that you do not modify this register unless directed by Arm.

This register can be read using MRRC with the following syntax:

```
MRCC <syntax>
```

This register can be written using MCRR with the following syntax:

```
MCRR <syntax>
```

This syntax is encoded with the following settings in the instruction encoding:

<syntax>	coproc	opc1	CRm
p15, 9, <Rt>, <Rt2>, c15	1111	1001	1111

Accessibility

This register is accessible in software as follows:

<syntax>	Control			Accessibility			
	E2H	TGE	NS	EL0	EL1	EL2	EL3
p15, 9, <Rt>, <Rt2>, c15	x	x	0	-	-	n/a	RW
p15, 9, <Rt>, <Rt2>, c15	x	0	1	-	-	-	RW
p15, 9, <Rt>, <Rt2>, c15	x	1	1	-	n/a	-	RW

'n/a' Not accessible. The core cannot be executing at this Exception level, so this access is not possible.

Traps and enables

For a description of the prioritization of any generated exceptions, see *Exception priority order* in the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile* for exceptions taken to AArch32 state, and see *Synchronous exception prioritization* for exceptions taken to AArch64 state.

B1.22 CPUPSELR, CPU Private Selection Register

The CPUPSELR provides IMPLEMENTATION DEFINED configuration and control options for the core.

Bit field descriptions

CPUPSELR is a 32-bit register, and is part of the Implementation defined registers functional group.

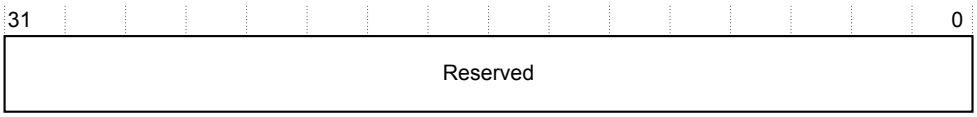

Figure B1-17 CPUPSELR bit assignments

Reserved, [31:0]

Reserved for Arm internal use.

Configurations

CPUPSELR is:

- Only accessible in Secure state.
- Mapped to the AArch64 CPUPSELR_EL3 register. See *B2.30 CPUPSELR_EL3, CPU Private Selection Register, EL3* on page B2-330.

Usage constraints

Accessing the CPUPSELR

Arm recommends that you write to this register after a powerup reset, before the MMU is enabled.

Writing to this register might cause UNPREDICTABLE behaviors. Therefore, Arm strongly recommends that you do not modify this register unless directed by Arm.

This register can be read using MRC with the following syntax:

```
MRC <syntax>
```

This register can be written using MCR with the following syntax:

```
MCR <syntax>
```

This syntax is encoded with the following settings in the instruction encoding:

<syntax>	coproc	opc1	CRn	CRm	Opc2
p15, 6, <Rt>, c15, c8, 0	1111	110	1111	1000	000

Accessibility

This register is accessible in software as follows:

<syntax>	Control			Accessibility			
	E2H	TGE	NS	EL0	EL1	EL2	EL3
p15, 6, <Rt>, c15, c8, 0	x	x	0	-	-	n/a	RW
p15, 6, <Rt>, c15, c8, 0	x	0	1	-	-	-	RW
p15, 6, <Rt>, c15, c8, 0	x	1	1	-	n/a	-	RW

'n/a' Not accessible. The core cannot be executing at this Exception level, so this access is not possible.

Traps and enables

For a description of the prioritization of any generated exceptions, see *Exception priority order* in the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile* for exceptions taken to AArch32 state, and see *Synchronous exception prioritization* for exceptions taken to AArch64 state.

B1.23 CPUPWRCTLR, CPU Power Control Register

The CPUPWRCTLR is a configuration register that gives indications to the external power controller.

Bit field descriptions

CPUPWRCTLR is a 32-bit register, and is part of the Implementation registers functional group.

This register resets to value `0x00000000`.

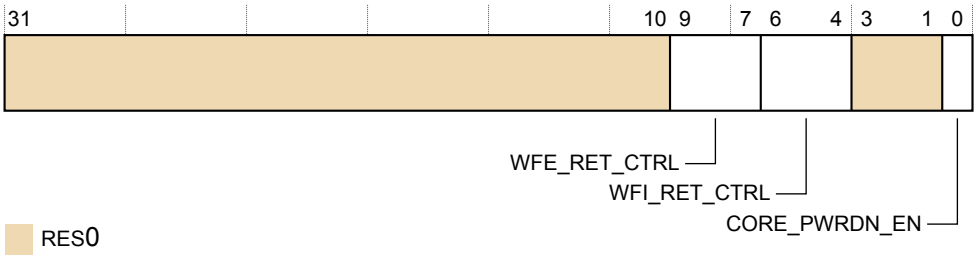

Figure B1-18 CPUPWRCTLR bit assignments

RES0, [31:10]

RES0 Reserved.

WFE_RET_CTRL, [9:7]

CPU WFE retention control:

`0b000` Disable the retention circuit. This is the default value, see *Table B1-7 CPUPWRCTLR Retention Control Field* on page B1-166 for more retention control options.

WFI_RET_CTRL, [6:4]

CPU WFI retention control:

`0b000` Disable the retention circuit. This is the default value, see *Table B1-7 CPUPWRCTLR Retention Control Field* on page B1-166 for more retention control options.

RES0, [3:1]

RES0 Reserved.

CORE_PWRDN_EN, [0]

Indicates to the power controller if the CPU wants to power down when it enters WFI state.

`0b0` No power down requested.

`0b1` A power down is requested.

Table B1-7 CPUPWRCTLR Retention Control Field

Encoding	Note	Minimum Delay before retention 50MHz – 10MHz
`000`	Disable the retention circuit.	Default Condition.
`001`	2 Architectural Timer ticks are required before retention entry.	40ns – 200ns
`010`	8 Architectural Timer ticks are required before retention entry.	160ns – 800ns
`011`	32 Architectural Timer ticks are required before retention entry.	640ns – 3,200ns
`100`	64 Architectural Timer ticks are required before retention entry.	1,280ns – 6,400ns
`101`	128 Architectural Timer ticks are required before retention entry.	2,560ns – 12,800ns

Table B1-7 CPUPWRCTLR Retention Control Field (continued)

Encoding	Note	Minimum Delay before retention 50MHz – 10MHz
110	256 Architectural Timer ticks are required before retention entry.	5,120ns – 25,600ns
111	512 Architectural Timer ticks are required before retention entry.	10,240ns – 51,200ns

Configurations

CPUPWRCTLR is architecturally mapped to AArch64 register CPUPWRCTLR_EL1. See *B2.31 CPUPWRCTLR_EL1, Power Control Register, EL1* on page B2-332.

Usage constraints

Accessing the CPUPWRCTLR

This register can be read using MRC with the following syntax:

```
MRC <syntax>
```

This register can be written using MCR with the following syntax:

```
MCR <syntax>
```

This syntax is encoded with the following settings in the instruction encoding:

<syntax>	coproc	opc1	CRn	CRm	opc2
p15, 0, <Rt>, c15, c2, 7	1111	000	1111	0010	111

Accessibility

This register is accessible in software as follows:

<syntax>	Control			Accessibility			
	E2H	TGE	NS	EL0	EL1	EL2	EL3
p15, 0, <Rt>, c15, c2, 7	x	x	0	-	RW	n/a	RW
p15, 0, <Rt>, c15, c2, 7	x	0	1	-	RW	RW	RW
p15, 0, <Rt>, c15, c2, 7	x	1	1	-	n/a	RW	RW

'n/a' Not accessible. The PE cannot be executing at this Exception level, so this access is not possible.

Traps and enables

For a description of the prioritization of any generated exceptions, see *Exception priority order* in the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile* for exceptions taken to AArch32 state, and see *Synchronous exception prioritization* for exceptions taken to AArch64 state.

Write access to this register from EL1 or EL2 depends on the value of bit[7] of ACTLR_EL2, ACTLR_EL3, ACTLR (S), and HACTLR.

B1.24 CSSELR, Cache Size Selection Register

The CSSELR selects the current CCSIDR by specifying:

- The required cache level.
- The cache type, either instruction or data cache.

For details of the CCSIDR, see *B1.12 CCSIDR, Cache Size ID Register* on page B1-144.

Bit field descriptions

CSSELR is a 32-bit register, and is part of the Identification registers functional group.

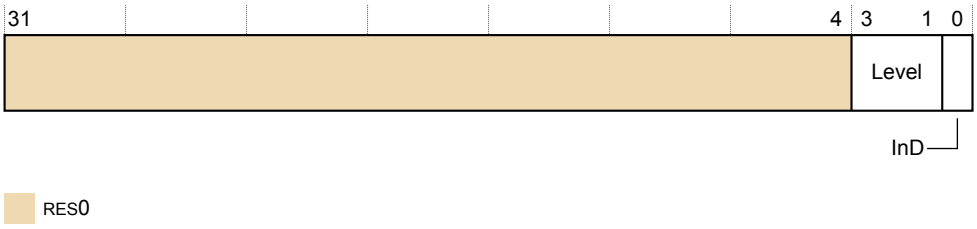

Figure B1-19 CSSELR bit assignments

RES0, [31:4]

RES0 Reserved.

Level, [3:1]

Cache level of required cache:

`0b000` L1.

`0b001` L2.

`0b010` L3. Only if L3 exists.

The combination of Level=`0b001` and InD=`0b1` is reserved.

The combinations of Level and InD for `0b0100` to `0b1111` are reserved.

InD, [0]

Instruction not Data bit:

`0b0` Data or unified cache.

`0b1` Instruction cache.

The combination of Level=`0b001` or Level=`0b010` and InD=`0b1` is reserved.

The combinations of Level and InD for `0b0100` to `0b1111` are reserved.

Configurations

CSSELR (NS) is architecturally mapped to AArch64 register CSSELR_EL1. See *B2.32 CSSELR_EL1, Cache Size Selection Register, EL1* on page B2-334.

If a cache level is missing but CSSELR selects this level, then CCSIDR is L1 cache as CSSERL is RES0 for all bits when programmed with a cache level which does not exist.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

B1.25 CTR, Cache Type Register

The CTR provides information about the architecture of the caches.

Bit field descriptions

CTR is a 32-bit register, and is part of the Identification registers functional group.

This register is Read Only.

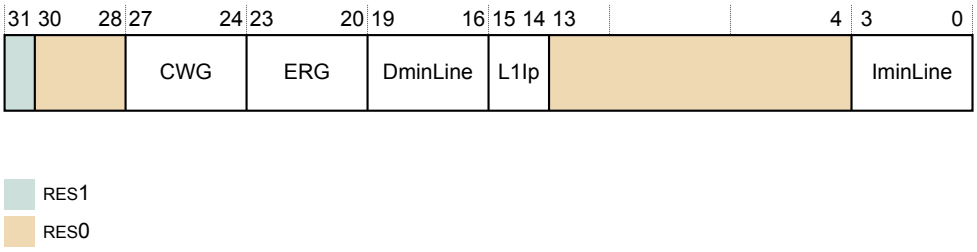

Figure B1-20 CTR bit assignments

RES1, [31]

RES1 Reserved.

RES0, [30:28]

RES0 Reserved.

CWG, [27:24]

Cache Write-Back 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:

`0b0100` Cache Write-Back granule size is 16 words.

ERG, [23:20]

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:

`0b0100` Exclusive reservation granule size is 16 words.

DminLine, [19:16]

Log_2 of the number of words in the smallest cache line of all the data and unified caches that the core controls:

`0b0100` Smallest data cache line size is 16 words.

L1Ip, [15:14]

Instruction cache policy. Indicates the indexing and tagging policy for the L1 Instruction cache:

`0b10` *Virtually Indexed Physically Tagged* (VIPT).

RES0, [13:4]

RES0 Reserved.

IminLine, [3:0]

Log_2 of the number of words in the smallest cache line of all the instruction caches that the core controls.

`0b0100` Smallest instruction cache line size is 16 words.

Configurations

CTR is architecturally mapped to AArch64 register CTR_EL0. See *B2.33 CTR_EL0, Cache Type Register, EL0* on page B2-335.

There is one copy of this register that is used in both Secure and Non-secure states.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

B1.26 DFSR, Data Fault Status Register

The DFSR holds status information about the last data fault.

Bit field descriptions

DFSR is a 32-bit register, and is part of the Exception and fault handling registers functional group.

There are two formats for this register. The current translation table format determines which format of the register is used.

- *B1.26.1 DFSR with Short-descriptor translation table format* on page B1-171.
- *B1.26.2 DFSR with Long-descriptor translation table format* on page B1-171.

Configurations

DFSR (NS) is architecturally mapped to AArch64 register ESR_EL1. See *B2.47 ESR_EL1, Exception Syndrome Register, EL1* on page B2-352.

DFSR (S) is architecturally mapped to AArch64 register ESR_EL3. See *B2.49 ESR_EL3, Exception Syndrome Register, EL3* on page B2-354.

If EL3 is using AArch32, there are separate Secure and Non-secure instances of this register.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

This section contains the following subsections:

- *B1.26.1 DFSR with Short-descriptor translation table format* on page B1-171.
- *B1.26.2 DFSR with Long-descriptor translation table format* on page B1-171.

B1.26.1 DFSR with Short-descriptor translation table format

DFSR has a specific format when using the Short-descriptor translation table format.

The following figure shows the DFSR bit assignments when using the Short-descriptor translation table format.

When TTBCR.EAE==0:

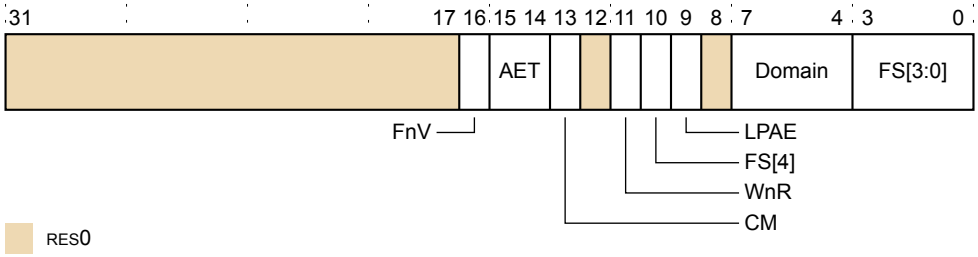

Figure B1-21 DFSR bit assignments for Short-descriptor translation table format

AET, [15:14]

Asynchronous Error Type. Describes the state of the PE after taking an asynchronous Data Abort exception. The value is:

`0b00` *Uncorrected error* (UC) or uncategorized error.

ExT, [12]

RES0 Reserved. This bit is unused.

B1.26.2 DFSR with Long-descriptor translation table format

DFSR has a specific format when using the Long-descriptor translation table format.

The following figure shows the DFSR bit assignments when using the Long-descriptor translation table format.

When TTBCR.EAE==0:

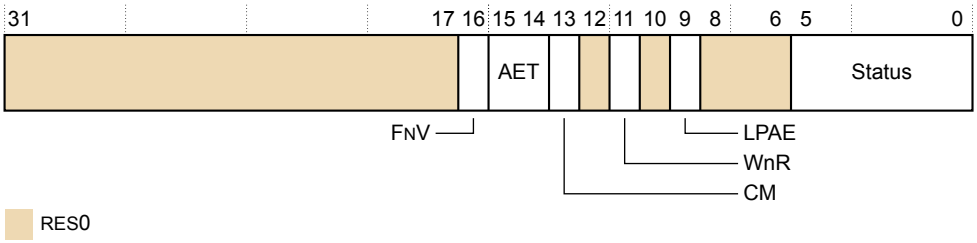

Figure B1-22 DFSR bit assignments for Long-descriptor translation table format

AET, [15:14]

Asynchronous Error Type. Describes the state of the PE after taking an asynchronous Data Abort exception. The value is:

`0b00` *Uncorrected error* (UC) or uncategorized error.

ExT, [12]

RES0 Reserved. This bit is unused.

B1.27 DISR, Deferred Interrupt Status Register

DISR records that an SError interrupt has been consumed by an `ESB` instruction.

Bit field descriptions

DISR is a 32-bit register, and is part of the RAS registers group.

There are three formats for this register. The current translation table format determines which format of the register is used.

- When written at EL1 using Short-descriptor format. See *B1.27.1 DISR with Short-descriptor translation table format* on page B1-173.
- When written at EL1 using Long-descriptor format. See *B1.27.2 DISR with Long-descriptor translation table format* on page B1-174.
- When written at EL2. See *B1.27.3 DISR at EL2* on page B1-175.

Configurations

AArch32 register DISR is architecturally mapped to AArch64 register DISR_EL1. See *B2.35 DISR_EL1, Deferred Interrupt Status Register, EL1* on page B2-338.

There is one instance of DISR that is used in both Secure and Non-secure states.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

This section contains the following subsections:

- *B1.27.1 DISR with Short-descriptor translation table format* on page B1-173.
- *B1.27.2 DISR with Long-descriptor translation table format* on page B1-174.
- *B1.27.3 DISR at EL2* on page B1-175.

B1.27.1 DISR with Short-descriptor translation table format

DISR has a specific format when written at EL1 using the Short-descriptor translation table format.

The following figure shows the DISR bit assignments when using the Short-descriptor translation table format.

When TTBCR.EAE==0:

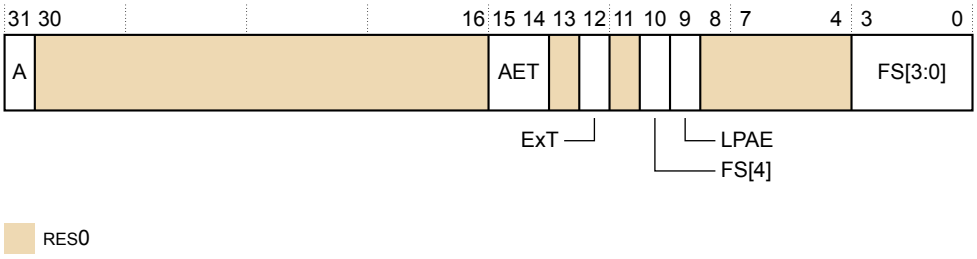

Figure B1-23 DISR bit assignments for Short-descriptor translation table format

A, [31]

Set to 1 when ESB defers an asynchronous SError interrupt.

RES0, [30:16]

RES0 Reserved.

AET, [15:14]

Asynchronous Error Type. Describes the state of the core after taking an asynchronous Data Abort exception. The value is:

RES0. The core is always in *Uncontainable* (UC) state when an SEI is signaled.

RES0, [13]

RES0 Reserved.

Ext, [12]

External Abort Type. This bit is defined as RES0.

RES0, [11]

RES0 Reserved.

FS[4], [10]

Fault Status Code. See the description of DFSR.FS in the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile* for an SError interrupt.

LPAE, [9]

On taking a Data Abort exception, this bit is set as follows:

`0` Using the Short-descriptor translation table formats.

Hardware does not interpret this bit to determine the behavior of the memory system, and therefore software can set this bit to 0 or 1 without affecting operation.

RES0, [8:4]

RES0 Reserved.

FS[3:0], [3:0]

Fault Status bits. This field indicates the type of exception generated. See the description of DFSR.FS in the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile* for an SError interrupt.

B1.27.2 DISR with Long-descriptor translation table format

DISR has a specific format when written at EL1 using the Long-descriptor translation table format.

The following figure shows the DISR bit assignments when using the Long-descriptor translation table format.

When TTBCR.EAE==1:

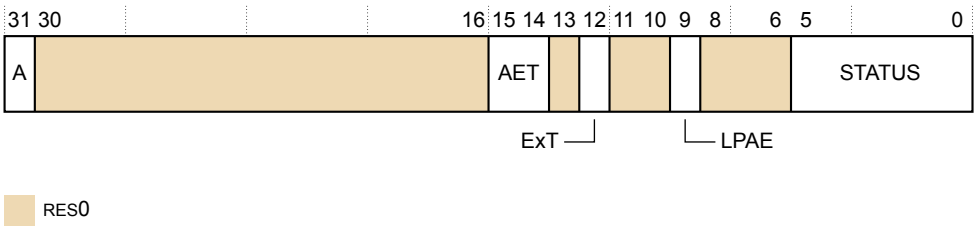

Figure B1-24 DISR bit assignments for Long-descriptor translation table format

A, [31]

Set to 1 when ESB defers an asynchronous SError interrupt.

RES0, [30:16]

RES0 Reserved.

AET, [15:14]

Asynchronous Error Type. Describes the state of the core after taking an asynchronous Data Abort exception. The value is:

RES0. The core is always in *Uncontainable* (UC) state when an SEI is signaled.

RES0, [13]

RES0 Reserved.

Ext, [12]
External Abort Type. This bit is defined as RES0.

RES0, [11:10]

RES0 Reserved.

LPAE, [9]
On taking a Data Abort exception, this bit is set as follows:

1 Using the Long-descriptor translation table formats.

Hardware does not interpret this bit to determine the behavior of the memory system, and therefore software can set this bit to 0 or 1 without affecting operation.

RES0, [8:6]

RES0 Reserved.

Status, [5:0]
Fault Status Code. This field indicates the type of exception generated. See the DFSR.DFSC in the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile* for an SError interrupt.

B1.27.3 DISR at EL2

DISR has a specific format when written at EL2.

The following figure shows the DISR bit assignments when written at EL2:

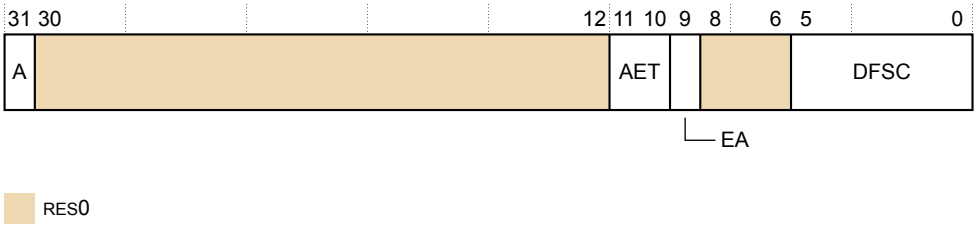

Figure B1-25 DISR bit assignments for Long-descriptor translation table format

A, [31]
Set to 1 when ESB defers an asynchronous SError interrupt. If the implementation does not include any synchronizable sources of SError interrupt, this bit is res0.

RES0, [30:12]

RES0 Reserved.

AET, [11:10]
Asynchronous Error Type. Describes the state of the core after taking the SError interrupt exception. Software might use the information in the syndrome registers to determine what recovery might be possible. The value is:

RES0. The core is always in *Uncontainable* (UC) state when and an SEI is signaled.

EA, [9]
External Abort Type. This bit is defined as RES0.

RES0, [8:6]

RES0 Reserved.

DFSC, [5:0]
Fault Status Code. This field indicates the type of exception generated. See the description of HSR.DFSC in the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile* for an SError interrupt.

B1.28 ERRIDR, Error ID Register

The ERRIDR defines the number of error records that can be accessed through the Error Record system registers.

Bit field descriptions

ERRIDR is a 32-bit register, and is part of the *Reliability, Availability, Serviceability* (RAS) registers functional group.

This register is Read Only.

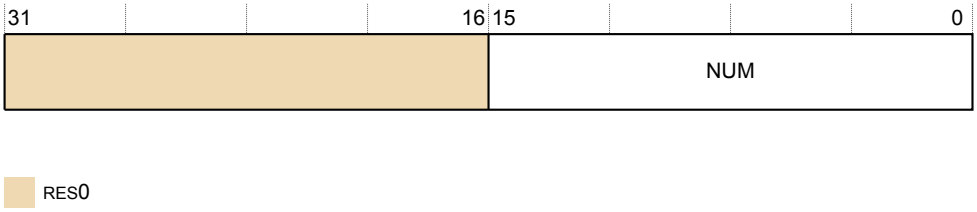

Figure B1-26 ERRIDR bit assignments

RES0, [31:16]

RES0 Reserved.

NUM, [15:0]

Number of records that can be accessed through the Error Record system registers.

`0x0002` Two records present.

Configurations

ERRIDR is architecturally mapped to AArch64 register ERRIDR_EL1. See *B2.36 ERRIDR_EL1, Error ID Register, EL1* on page B2-339.

There is one copy of this register that is used in both Secure and Non-secure states.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

B1.29 ERRSELR, Error Record Select Register

The ERRSELR selects which error record should be accessed through the Error Record system registers. This register is not reset on a warm reset.

Bit field descriptions

ERRSELR is a 32-bit register, and is part of the *Reliability, Availability, Serviceability* (RAS) registers functional group.

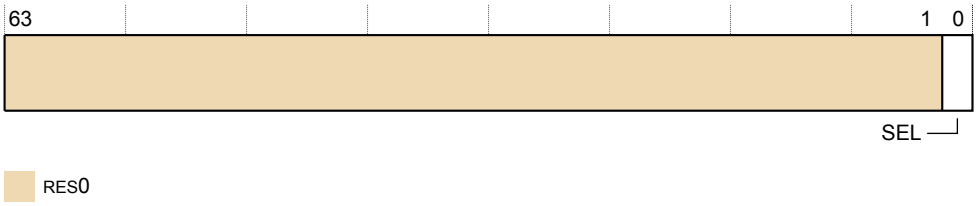

Figure B1-27 ERRSELR bit assignments

RES0, [31:1]

RES0 Reserved.

SEL, [0]

Selects the record accessed through the Error Record system registers.

0 Select record 0 containing errors from level-1 and level-2 RAMs located in the Cortex-A75 core.

1 Select record 1 containing errors from level-3 RAMs located in the DSU.

Configurations

ERRSELR is architecturally mapped to AArch64 register ERRSELR_EL1. See *B2.37 ERRSELR_EL1, Error Record Select Register, EL1* on page B2-340.

There is one copy of this register that is used in both Secure and Non-secure states.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

B1.30 ERXADDR, Selected Error Record Address Register

The ERXADDR accesses bits [31:0] of the ERR<n>ADRR address register for the error record selected by ERRSELR.SEL.

If ERRSELR.SEL==0, then ERXADDR accesses the ERR0ADDR register of the core error record. See *B3.2 ERR0ADDR, Error Record Address Register* on page B3-440.

If ERRSELR.SEL==1, then ERXADDR accesses the ERR1ADDR register of the DSU error record. See the *Arm® DynamIQ™ Shared Unit Technical Reference Manual*.

B1.31 ERXADDR2, Selected Error Record Address Register 2

The ERXADDR2 accesses bits [63:32] of the ERR<n>ADRR address register for the error record selected by ERRSELR.SEL.

If ERRSELR.SEL==0, then ERXADDR2 accesses the ERR0ADDR register of the core error record. See *B3.2 ERR0ADDR, Error Record Address Register* on page B3-440.

If ERRSELR.SEL==1, then ERXADDR2 accesses the ERR1ADDR register of the DSU error record. See the *Arm® DynamIQ™ Shared Unit Technical Reference Manual*.

B1.32 ERXCTLR, Selected Error Record Control Register

The ERXCTLR accesses bits [31:0] of the ERR<n>CTLR control register for the error record selected by ERRSELR.SEL.

If ERRSELR.SEL==0, then ERXCTLR accesses the ERR0CTLR register of the core error record. See *B3.3 ERR0CTLR, Error Record Control Register* on page B3-441.

If ERRSELR.SEL==1, then ERXCLTR accesses the ERR1CTLR register of the DSU error record. See the *Arm® DynamIQ™ Shared Unit Technical Reference Manual*.

B1.33 ERXCTLR2, Selected Error Record Control Register 2

The ERXCTLR2 accesses bits [62:32] of the ERR<n>CTLR control register for the error record selected by ERRSELR.SEL.

If ERRSELR.SEL==0, then ERXCTLR2 accesses the ERR0CTLR register of the core error record. See *B3.3 ERR0CTLR, Error Record Control Register* on page B3-441.

If ERRSELR.SEL==1, then ERXCLTR2 accesses the ERR1CTLR register of the DSU error record. See the *Arm® DynamIQ™ Shared Unit Technical Reference Manual*.

B1.34 ERXFR, Selected Error Record Feature Register

Register ERXFR accesses bits [31:0] of the ERR<n>FR feature register for the error record selected by ERRSELR.SEL.

If ERRSELR.SEL==0, then ERXFR accesses the ERR0FR register of the core error record. See *B3.4 ERR0FR, Error Record Feature Register* on page B3-443.

If ERRSELR.SEL==1, then ERXCLTR accesses the ERR1FR register of the DSU error record. See the *Arm® DynamIQ™ Shared Unit Technical Reference Manual*.

B1.35 ERXFR2, Selected Error Record Feature Register 2

Register ERXFR2 accesses bits [63:32] of the ERR<n>FR feature register for the error record selected by ERRSELR.SEL.

If ERRSELR.SEL==0, then ERXFR2 accesses the ERR0FR register of the core error record. See *B3.4 ERR0FR, Error Record Feature Register* on page B3-443.

If ERRSELR.SEL==1, then ERXCLTR accesses the ERR1FR register of the DSU error record. See the *Arm® DynamIQ™ Shared Unit Technical Reference Manual*.

B1.36 ERXMISC0, Selected Error Miscellaneous Register 0

Register ERXMISC0 accesses bits [31:0] of the ERR<n>MISC0 control register for the error record selected by ERRSELR.SEL.

If ERRSELR.SEL==0, then ERXMISC0 accesses bits [31:0] of the ERR0MISC0 register for the core error record. See *B3.5 ERR0MISC0, Error Record Miscellaneous Register 0* on page B3-445.

If ERRSELR.SEL==1, then ERXMISC0 accesses bits [31:0] of the ERR1MISC0 register for the DSU error record. See the *Arm® DynamIQ™ Shared Unit Technical Reference Manual*.

B1.37 ERXMISC1, Selected Error Miscellaneous Register 1

Register ERXMISC1 accesses bits [63:32] of the ERR<n>MISC0 miscellaneous register 0 for the error record selected by ERRSELR.SEL.

If ERRSELR.SEL==0, then ERXMISC1 accesses the ERR0MISC0[63:32] register of the core error record. See *B3.5 ERR0MISC0, Error Record Miscellaneous Register 0* on page B3-445.

If ERRSELR.SEL==1, then ERXMISC1 accesses the ERR1MISC0[63:32] register of the DSU error record. See the *Arm® DynamIQ™ Shared Unit Technical Reference Manual*.

B1.38 ERXMISC2, Selected Error Record Miscellaneous Register 2

Register ERXMISC2 accesses bits [31:0] of the ERR<n>MISC1 miscellaneous register 1 for the error record selected by ERRSELR.SEL.

If ERRSELR.SEL==0, then ERXMISC2 accesses the ERR0MISC1[31:0] register of the core error record. See *B3.6 ERR0MISC1, Error Record Miscellaneous Register 1* on page B3-447.

If ERRSELR.SEL==1, then ERXMISC2 accesses the ERR1MISC1[31:0] register of the DSU error record. See the *Arm® DynamIQ™ Shared Unit Technical Reference Manual*.

B1.39 ERXMISC3, Selected Error Record Miscellaneous Register 3

Register ERXMISC3 accesses bits [63:32] of the ERR<n>MISC1 miscellaneous register 1 for the error record selected by ERRSELR.SEL.

If ERRSELR.SEL==0, then ERXMISC3 accesses the ERR0MISC1[63:32] register of the core error record. See *B3.6 ERR0MISC1, Error Record Miscellaneous Register 1* on page B3-447.

If ERRSELR.SEL==1, then ERXMISC3 accesses the ERR1MISC1[63:32] register of the DSU error record. See the *Arm® DynamIQ™ Shared Unit Technical Reference Manual*.

B1.40 ERXPFGCDNR, Selected Error Pseudo Fault Generation Count Down Register

Register ERXPFGCDNR accesses the ERR<n>PFGCNDR register for the error record selected by ERRSELR.SEL.

If ERRSELR.SEL==0, then ERXPFGCDNR accesses the ERR0PFGCDNR register of the core error record. See *B3.7 ERR0PFGCDNR, Error Pseudo Fault Generation Count Down Register* on page B3-448.

If ERRSELR.SEL==1, then ERXPFGCDNR accesses the ERR1PFGCDNR register of the DSU error record. See the *Arm® DynamIQ™ Shared Unit Technical Reference Manual*.

Configurations

ERXPFGCDNR is architecturally mapped to AArch64 register ERXPFGCDNR_EL1. See *B2.43 ERXPFGCDNR_EL1, Selected Error Pseudo Fault Generation Count Down Register, EL1* on page B2-346.

Accessing the ERXPFGCDNR

This register can be read using MRC with the following syntax:

```
MRC <syntax>
```

This register can be written using MCR with the following syntax:

```
MCR <syntax>
```

This syntax is encoded with the following settings in the instruction encoding:

<syntax>	coproc	opc1	CRn	CRm	opc2
p15, 0, <Rt>, c15, c2, 2	1111	000	1111	0010	010

Accessibility

This register is accessible in software as follows:

<syntax>	Control			Accessibility			
	E2H	TGE	NS	EL0	EL1	EL2	EL3
p15, 0, <Rt>, c15, c2, 2	x	x	0	-	RW	n/a	RW
p15, 0, <Rt>, c15, c2, 2	x	0	1	-	RW	RW	RW
p15, 0, <Rt>, c15, c2, 2	x	1	1	-	n/a	RW	RW

'n/a' Not accessible. The PE cannot be executing at this Exception level, so this access is not possible.

Traps and enables

For a description of the prioritization of any generated exceptions, see *Exception priority order* in the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile* for exceptions taken to AArch32 state, and see *Synchronous exception prioritization* for exceptions taken to AArch64 state. Subject to these prioritization rules, the following traps and enables are applicable when accessing this register.

ERXPFGCDNR is accessible at EL3 and can be accessible at EL1 and EL2 depending on the value of bit[5] in ACTLR(S) and HACTLR. See *B1.5 ACTLR, Auxiliary Control Register* on page B1-136 and *B1.46 HACTLR, Hyp Auxiliary Control Register* on page B1-196.

If EL2 or EL3 are in AArch64, then access to lower exception levels is controlled by ACTLR_EL2 or ACTLR_EL3. See *B2.6 ACTLR_EL2, Auxiliary Control Register, EL2* on page B2-294 and *B2.7 ACTLR_EL3, Auxiliary Control Register, EL3* on page B2-296.

ERXPFGCDNR is UNDEFINED at EL0.

If ERXPFGCDNR is accessible at EL1, EL2 is using AArch32, and HCR.TERR == 1, then direct reads and writes of ERXPFGCDNR at Non-secure EL1 generate a Trap exception to EL2.

If ERXPFGCDNR is accessible at EL1, EL2 is using AArch64, and HCR_EL2.TERR == 1, then direct reads and writes of ERXPFGCDNR at Non-secure EL1 generate a Trap exception to EL2.

If ERXPFGCDNR is accessible at EL1 or EL2, EL3 is using AArch32, and SCR.TERR == 1, then direct reads and writes of ERXPFGCDNR at EL1 generate a Trap exception to EL3.

If ERXPFGCDNR is accessible at EL1 or EL2, EL3 is using AArch64, and SCR_EL3.TERR == 1, then direct reads and writes of ERXPFGCDNR at EL1 and EL2 generate a Trap exception to EL3.

B1.41 ERXPFGCTLR, Selected Error Pseudo Fault Generation Control Register

Register ERXPFGCTLR accesses bits [31:0] of the ERR<n>PFGCTLR register for the error record selected by ERRSELR.SEL.

If ERRSELR.SEL==0, then ERXPFGCTLR accesses the ERR0PFGCTLR register of the core error record. See *B3.8 ERR0PFGCTLR, Error Pseudo Fault Generation Control Register* on page B3-449.

If ERRSELR.SEL==1, then ERXPFGCTLR accesses the ERR1PFGCTLR register of the DSU error record. See the *Arm® DynamIQ™ Shared Unit Technical Reference Manual*.

Configurations

ERXPFGCTLR is architecturally mapped to AArch64 register ERXPFGCTLR_EL1. See *B2.44 ERXPFGCTLR_EL1, Selected Error Pseudo Fault Generation Control Register, EL1* on page B2-348.

Accessing the ERXPFGCTLR

This register can be read using MRC with the following syntax:

```
MRC <syntax>
```

This register can be written using MCR with the following syntax:

```
MCR <syntax>
```

This syntax is encoded with the following settings in the instruction encoding:

<syntax>	coproc	opc1	CRn	CRm	opc2
p15, 0, <Rt>, c15, c2, 1	1111	000	1111	0010	001

Accessibility

This register is accessible in software as follows:

<syntax>	Control			Accessibility			
	E2H	TGE	NS	EL0	EL1	EL2	EL3
p15, 0, <Rt>, c15, c2, 1	x	x	0	-	RW	n/a	RW
p15, 0, <Rt>, c15, c2, 1	x	0	1	-	RW	RW	RW
p15, 0, <Rt>, c15, c2, 1	x	1	1	-	n/a	RW	RW

'n/a' Not accessible. The PE cannot be executing at this Exception level, so this access is not possible.

Traps and enables

For a description of the prioritization of any generated exceptions, see *Exception priority order* in the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile* for exceptions taken to AArch32 state, and see *Synchronous exception prioritization* for exceptions taken to AArch64 state. Subject to these prioritization rules, the following traps and enables are applicable when accessing this register.

ERXPFGCTLR is accessible at EL3 and can be accessible at EL1 and EL2 depending on the value of bit[5] in ACTLR(S) and HACTLR. See *B1.5 ACTLR, Auxiliary Control Register* on page B1-136 and *B1.46 HACTLR, Hyp Auxiliary Control Register* on page B1-196.

If EL2 or EL3 are in AArch64, then access to lower exception levels is controlled by ACTLR_EL2 or ACTLR_EL3. See *B2.6 ACTLR_EL2, Auxiliary Control Register, EL2* on page B2-294 and *B2.7 ACTLR_EL3, Auxiliary Control Register, EL3* on page B2-296.

ERXPFGCTLR is UNDEFINED at EL0.

If ERXPFGCTLR is accessible at EL1, EL2 is using AArch32, and HCR.TERR == 1, then direct reads and writes of ERXPFGCTLR at Non-secure EL1 generate a Trap exception to EL2.

If ERXPFGCTLR is accessible at EL1, EL2 is using AArch64, and HCR_EL2.TERR == 1, then direct reads and writes of ERXPFGCTLR at Non-secure EL1 generate a Trap exception to EL2.

If ERXPFGCTLR is accessible at EL1 or EL2, EL3 is using AArch32, and SCR.TERR == 1, then direct reads and writes of ERXPFGCTLR at EL1 generate a Trap exception to EL3.

If ERXPFGCTLR is accessible at EL1 or EL2, EL3 is using AArch64, and SCR_EL3.TERR == 1, then direct reads and writes of ERXPFGCTLR at EL1 and EL2 generate a Trap exception to EL3.

B1.42 ERXPFGFR, Selected Pseudo Fault Generation Feature Register

Register ERXPFGFR accesses bits [31:0] of the ERR<n>PFGFR register for the error record selected by ERRSELR.SEL.

If ERRSELR.SEL==0, then ERXPFGFR accesses the ERR0PFGFR register of the core error record. See *B3.9 ERR0PFGFR, Error Pseudo Fault Generation Feature Register* on page B3-451.

If ERRSELR.SEL==1, then ERXPFGFR accesses the ERR1PFGFR register of the DSU error record. See the *Arm® DynamIQ™ Shared Unit Technical Reference Manual*.

Configurations

ERXPFGFR is architecturally mapped to AArch64 register ERXPFGFR_EL1. See *B2.45 ERXPFGFR_EL1, Selected Pseudo Fault Generation Feature Register, EL1* on page B2-350.

Accessing the ERXPFGFR

This register can be read using MRC with the following syntax:

```
MRC <syntax>
```

This syntax is encoded with the following settings in the instruction encoding:

<syntax>	coproc	opc1	CRn	CRm	opc2
p15, 0, <Rt>, c15, c2, 0	1111	000	1111	0010	000

Accessibility

This register is accessible in software as follows:

<syntax>	Control			Accessibility			
	E2H	TGE	NS	EL0	EL1	EL2	EL3
p15, 0, <Rt>, c15, c2, 0	x	x	0	-	RO	n/a	RO
p15, 0, <Rt>, c15, c2, 0	x	0	1	-	RO	RO	RO
p15, 0, <Rt>, c15, c2, 0	x	1	1	-	n/a	RO	RO

'n/a' Not accessible. The PE cannot be executing at this Exception level, so this access is not possible.

Traps and enables

For a description of the prioritization of any generated exceptions, see *Exception priority order* in the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile* for exceptions taken to AArch32 state, and see *Synchronous exception prioritization* for exceptions taken to AArch64 state. Subject to these prioritization rules, the following traps and enables are applicable when accessing this register.
ERXPFGFR is UNDEFINED at EL0.

If ERXPFGFR is accessible at EL1, EL2 is using AArch32, and HCR.TERR == 1, then direct reads and writes of ERXPFGFR at Non-secure EL1 generate a Trap exception to EL2.

If ERXPFGFR is accessible at EL1, EL2 is using AArch64, and HCR_EL2.TERR == 1, then direct reads and writes of ERXPFGFR at Non-secure EL1 generate a Trap exception to EL2.

If ERXPFGFR is accessible at EL1 or EL2, EL3 is using AArch32, and SCR.TERR == 1, then direct reads and writes of ERXPFGFR at EL1 generate a Trap exception to EL3.

If ERXPFGFR is accessible at EL1 or EL2, EL3 is using AArch64, and SCR_EL3.TERR == 1, then direct reads and writes of ERXPFGFR at EL1 and EL2 generate a Trap exception to EL3.

B1.43 ERXSTATUS, Selected Error Record Primary Status Register

Register ERXSTATUS accesses the ERR<n>STATUS status register for the error record selected by ERRSELR.SEL.

If ERRSELR.SEL==0, then ERXSTATUS accesses the ERR0STATUS register of the core error record. See *B3.10 ERR0STATUS, Error Record Primary Status Register* on page B3-453.

If ERRSELR.SEL==1, then ERXSTATUS accesses the ERR1STATUS register of the DSU error record. See the *Arm® DynamIQ™ Shared Unit Technical Reference Manual*.

B1.44 FCSEIDR, FCSE Process ID Register

The FCSEIDR identifies whether the *Fast Context Switch Extension* (FCSE) is implemented.

Bit field descriptions

FCSEIDR is a 32-bit register, and is part of the Legacy feature registers functional group.

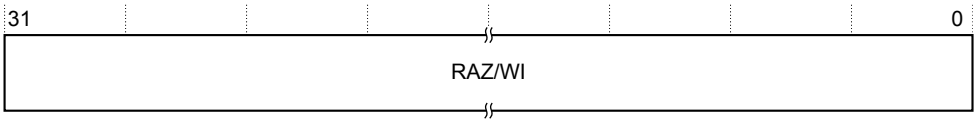

Figure B1-28 FCSEIDR bit assignments

RAZ/WI, [31:0]

Reserved, read-as-zero/write ignore.

Configurations

There is one instance of this register that is used in both Secure and Non-secure states.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

B1.45 HACR, Hyp Auxiliary Configuration Register

HACR controls trapping to Hyp mode of IMPLEMENTATION DEFINED aspects of Non-secure EL1 or EL0 operation. This register is not used in the Cortex-A75 core.

Bit field descriptions

HACR is a 32-bit register, and is part of the Virtualization registers functional group.

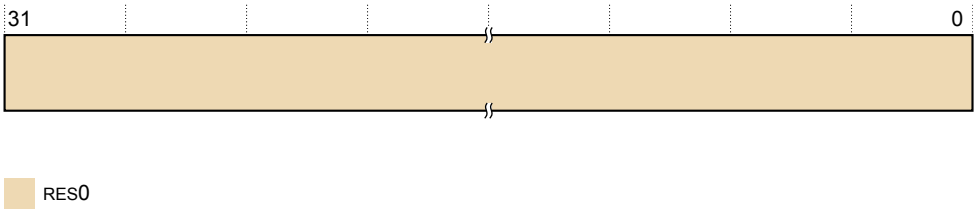

Figure B1-29 HACR bit assignments

RES0, [31:0]

RES0 Reserved.

Configurations

AArch32 System register HACR is architecturally mapped to AArch64 System register HACR_EL2. See *B2.50 HACR_EL2, Hyp Auxiliary Configuration Register, EL2* on page B2-355.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

B1.46 HACTLR, Hyp Auxiliary Control Register

The HACTLR controls IMPLEMENTATION DEFINED features of Hyp mode operation.

Bit field descriptions

HACTLR is a 32-bit register, and is part of:

- The Virtualization registers functional group.
- The Other system control registers functional group.
- The Implementation defined functional group.

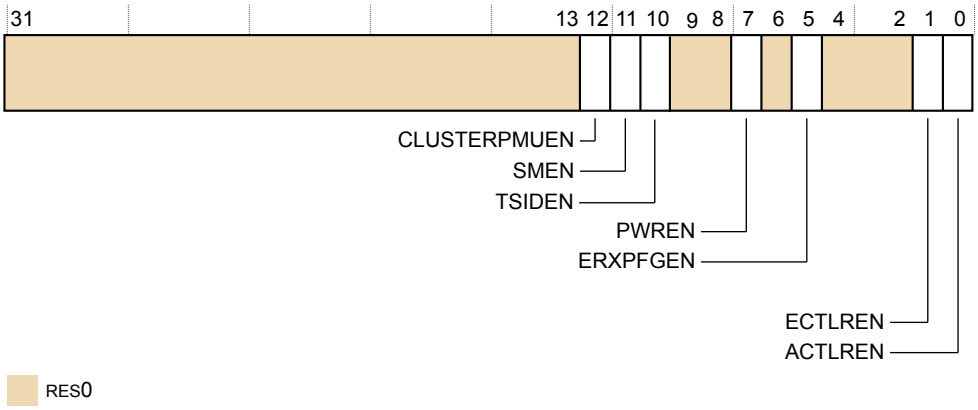

Figure B1-30 HACTLR bit assignments

RES0, [31:13]

RES0 Reserved.

CLUSTERPMUEN, [12]

Performance Management Registers enable. The value is:

0 CLUSTERPM* registers are not write accessible from a lower Exception level. This is the reset value.

1 CLUSTERPM* registers are write accessible from EL1 Non-secure if they are write accessible from EL2.

SMEN, [11]

Scheme Management Registers enable. The value is:

0 Registers CLUSTERTHREADSID, CLUSTERACPSID, CLUSTERSTASHSID, CLUSTERPARTCR, and CLUSTERBUSQOS are not write accessible from EL2. This is the reset value.

1 Registers controlled by the TSIDEN bit, CLUSTERACPSID, CLUSTERSTASHSID, CLUSTERPARTCR, and CLUSTERBUSQOS are write accessible from EL2.

TSIDEN, [10]

Thread Scheme ID Register enable. The possible values are:

0 Register CLUSTERTHREADSID is not accessible from EL1 nonsecure. This is the reset value.

1 Register CLUSTERTHREADSID is accessible from EL1 nonsecure if they are write accessible from EL2.

RES0, [9:8]

RES0 Reserved.

PWREN, [7]

Power Control Registers enable. The value is:

0 Registers CPUPWRCTLR, CLUSTERPWRCTLR, CLUSTERPWRDN, CLUSTERPWRSTAT, CLUSTERL3HIT and CLUSTERL3MISS are not write accessible from EL1. This is the reset value.

1 Registers CPUPWRCTLR, CLUSTERPWRCTLR, CLUSTERPWRDN, CLUSTERPWRSTAT, CLUSTERL3HIT and CLUSTERL3MISS are write accessible from EL1 Non-secure if they are write accessible from EL2.

RES0, [6]

RES0 Reserved.

ERXPFGEN, [5]

Error Record Registers enable. The value is:

0 ERXPFG* are not write accessible from EL1. This is the reset value.

1 ERXPFG* are write accessible from EL1 Non-secure if they are write accessible from EL2.

RES0, [4:2]

RES0 Reserved.

ECTLREN, [1]

Extended Control Registers enable. The value is:

0 CPUECTLR and CLUSTERECTLR are not write accessible from EL1. This is the reset value.

1 CPUECTLR and CLUSTERECTLR are write accessible from EL1 Non-secure if they are write accessible from EL2.

ACTLREN, [0]

Auxiliary Control Registers enable. The value is:

0 CPUACTLR, CPUACTLR2 and CLUSTERACTLR are not write accessible from EL1. This is the reset value.

1 CPUACTLR, CPUACTLR2 and CLUSTERACTLR are write accessible from EL1 Non-secure if they are write accessible from EL2.

Configurations

The HACTLR is architecturally mapped to the AArch64 ACTLR_EL2 register. See *B2.6 ACTLR_EL2, Auxiliary Control Register, EL2* on page B2-294.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

B1.47 HACTLR2, Hyp Auxiliary Control Register 2

The HACTLR2 Provides additional space to the HACTLR register to hold IMPLEMENTATION DEFINED trap functionality.

Bit field descriptions

HACTLR2 is a 32-bit register, and is part of:

- The Virtualization registers functional group.
- The Other system control registers functional group.
- The Implementation defined functional group.

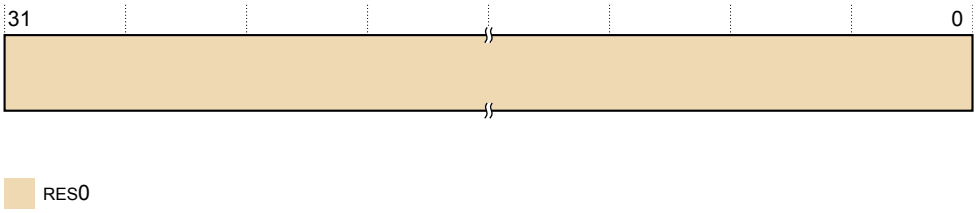

Figure B1-31 HACTLR2 bit assignments

RES0, [31:0]

RES0 Reserved.

Configurations

The HACTLR2 is architecturally mapped to the AArch64 ACTLR_EL2[63:32] register. See *B2.6 ACTLR_EL2, Auxiliary Control Register, EL2* on page B2-294.

RW fields in this register reset to architecturally UNKNOWN values.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

B1.48 HADFSR, Hyp Auxiliary Data Fault Status Syndrome Register

HADFSR provides additional IMPLEMENTATION DEFINED syndrome information for Data Abort exceptions taken to Hyp mode. This register is not used in the Cortex-A75 core.

Bit field descriptions

HADFSR is a 32-bit register, and is part of:

- The Virtualization registers functional group.
- The Exception and fault handling registers functional group.
- The Implementation defined functional group.

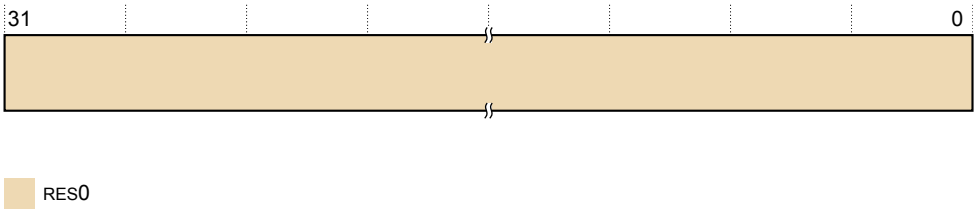

Figure B1-32 HADFSR bit assignments

RES0, [31:0]

RES0 Reserved.

Configurations

AArch32 System register HADFSR is architecturally mapped to AArch64 System register AFSR0_EL2. See *B2.9 AFSR0_EL2, Auxiliary Fault Status Register 0, EL2* on page B2-299.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

B1.49 HAIFSR, Hyp Auxiliary Instruction Fault Status Syndrome Register

HAIFSR provides additional IMPLEMENTATION DEFINED syndrome information for Prefetch Abort exceptions taken to Hyp mode. This register is not used in the Cortex-A75 core.

Bit field descriptions

HAIFSR is a 32-bit register, and is part of:

- The Virtualization registers functional group.
- The Exception and fault handling registers functional group.
- The Implementation defined functional group.

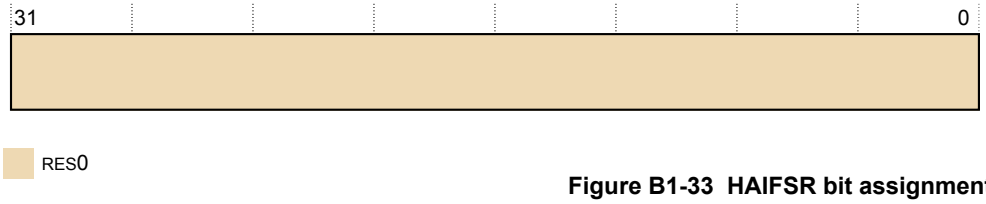

Figure B1-33 HAIFSR bit assignments

RES0, [31:0]

Reserved, RES0.

Configurations

AArch32 System register HAIFSR is architecturally mapped to AArch64 System register AFSR1_EL2. See *B2.12 AFSR1_EL2, Auxiliary Fault Status Register 1, EL2* on page B2-302.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

B1.50 HAMAIR0, Hyp Auxiliary Memory Attribute Indirection Register 0

HAMAIR0 provides additional IMPLEMENTATION DEFINED memory attributes for the memory attribute encodings defined by HMAIR0. These implementation defined attributes can only provide additional qualifiers for the memory attribute encodings, and cannot change the memory attributes defined in HMAIR0. This register is not used in the Cortex-A75 core.

Bit field descriptions

HAMAIR0 is a 32-bit register, and is part of:

- The Virtualization registers functional group.
- The Virtual memory control registers functional group.
- The Implementation defined functional group.

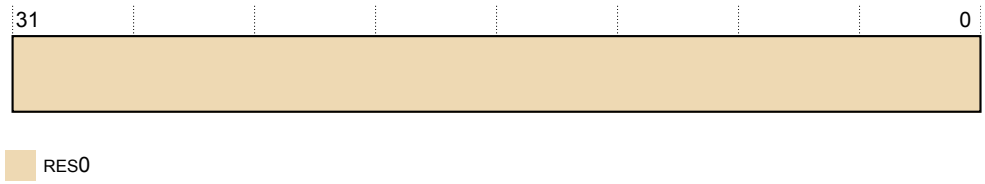

Figure B1-34 HAMAIR0 bit assignments

RES0, [31:0]

Reserved, RES0.

Configurations

AArch32 System register HAMAIR0 is architecturally mapped to AArch64 System register AMAIR_EL2[31:0]. See *B2.16 AMAIR_EL2, Auxiliary Memory Attribute Indirection Register, EL2* on page B2-306.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

B1.51 HAMAIR1, Hyp Auxiliary Memory Attribute Indirection Register 1

HAMAIR1 provides additional IMPLEMENTATION DEFINED memory attributes for the memory attribute encodings defined by HMAIR1. These implementation defined attributes can only provide additional qualifiers for the memory attribute encodings, and cannot change the memory attributes defined in HMAIR1. This register is not used in the Cortex-A75 core.

Bit field descriptions

HAMAIR1 is a 32-bit register, and is part of:

- The Virtualization registers functional group.
- The Virtual memory control registers functional group.
- The Implementation defined functional group.

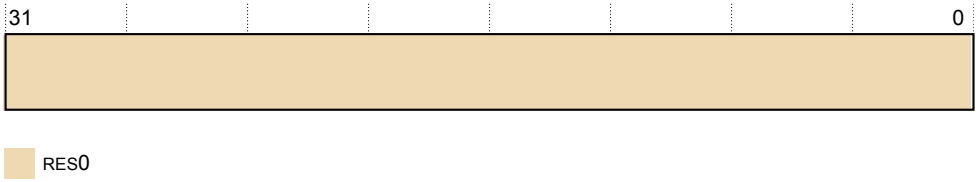

Figure B1-35 HAMAIR1 bit assignments

RES0, [31:0]

Reserved, RES0.

Configurations

AArch32 System register HAMAIR1 is architecturally mapped to AArch64 System register AMAIR_EL2[63:32]. See *B2.16 AMAIR_EL2, Auxiliary Memory Attribute Indirection Register, EL2* on page B2-306.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

B1.52 HCR, Hyp Configuration Register

The HCR provides configuration controls for virtualization, including defining whether various Non-secure operations are trapped to Hyp mode.

Bit field descriptions

HCR is a 32-bit register, and is part of the Virtualization registers functional group.

This register resets to value `0x00000002`.

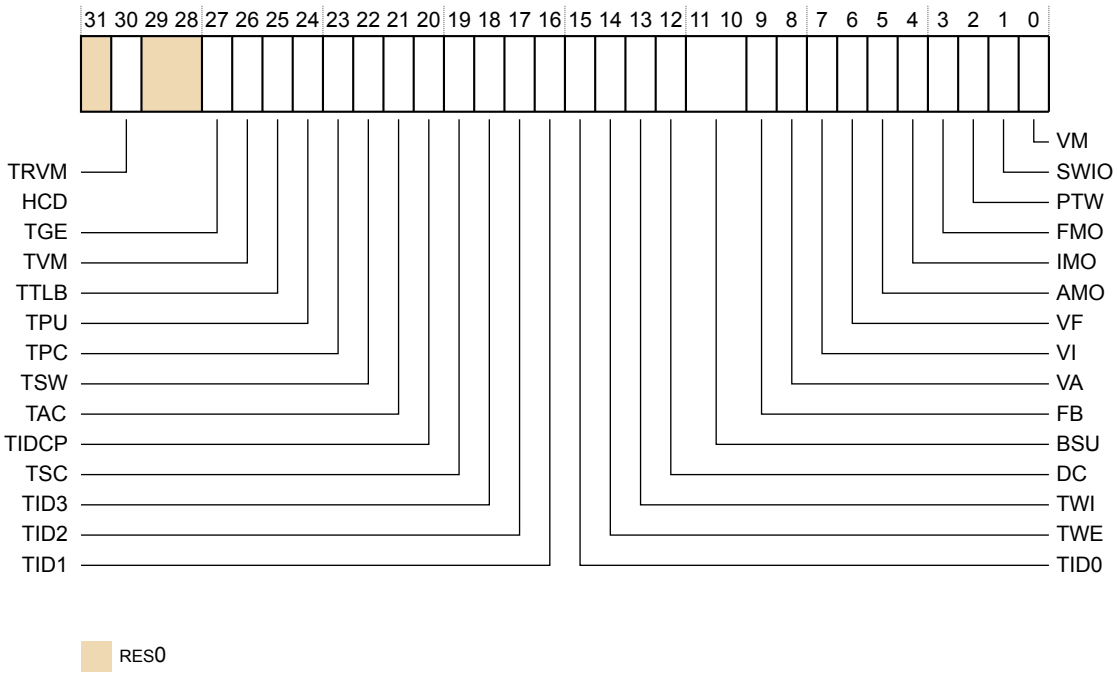

Figure B1-36 HCR bit assignments

RES0, [31]

RES0 Reserved.

RES0, [29:28]

RES0 Reserved.

TGE, [27]

Trap General Exceptions. If this bit is set, and SCR_EL3.NS is set, then:
All exceptions that would be routed to EL1 are routed to EL2.

- The SCTLR.M bit is treated as 0 regardless of its actual state, other than for reading the bit.
- The HCR.FMO, IMO, and AMO bits are treated as 1 regardless of their actual state, other than for reading the bits.
- All virtual interrupts are disabled.
- An exception return to EL1 is treated as an illegal exception return.

The Cortex-A75 core does not support any implementation defined mechanisms for signaling virtual interrupts.

Additionally, if HCR.TGE is 1, the HDCR.{TDRA,TDOSA,TDA} bits are ignored and the core behaves as if they are set to 1, other than for the value read back from HDCR.

TSC, [19]

Trap SMC instruction. When this bit is set to 1, any attempt from a Non-secure EL1 state to execute an SMC instruction, that passes its condition check if it is conditional, is trapped to Hyp mode.

SWIO, [1]

Set/Way Invalidation Override. This bit is RES1.

Configurations

HCR is architecturally mapped to AArch64 register HCR_EL2[31:0]. See *B2.51 HCR_EL2, Hypervisor Configuration Register, EL2* on page B2-356.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

B1.53 HCR2, Hyp Configuration Register 2

The HCR2 provides additional configuration controls for virtualization.

Bit field descriptions

HCR2 is a 32-bit register, and is part of the Virtualization registers functional group.

This register resets to value `0x00000000`.

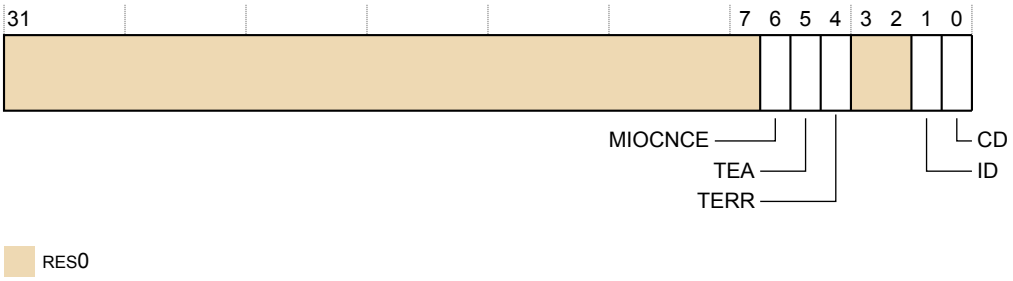

Figure B1-37 HCR2 bit assignments

MIOCNCE, [6]

Mismatched Inner/Outer Cacheable Non-Coherency Enable, for the Non-secure PL1&0 translation regime.

This bit is not implemented, RAZ/WI.

Configurations

HCR2 is architecturally mapped to AArch64 register HCR_EL2[63:32]. See *B2.51 HCR_EL2, Hypervisor Configuration Register, EL2* on page B2-356.

This register is accessible only at EL2 or EL3.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

B1.54 HSCTLR, Hyp System Control Register

The HSCTLR provides top level control of the system operation in Hyp mode.

This register provides Hyp mode control of features controlled by the Banked SCTLR bits, and shows the values of the non-Banked SCTLR bits.

Bit field descriptions

HSCTLR is a 32-bit register, and is part of:

- The Virtualization registers functional group.
- The Other system control registers functional group.

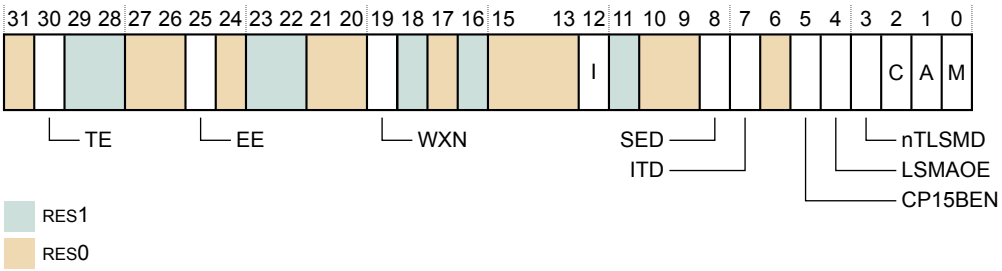

Figure B1-38 HSCTLR bit assignments

I, [12]

Instruction cache enable. This is an enable bit for instruction caches at EL2:

`0` Instruction caches disabled at EL2. If HSCTLR.M is set to 0, instruction accesses from stage 1 of the EL2 translation regime are to Normal memory, Outer Shareable, Inner Non-cacheable, Outer Non-cacheable. This is the reset value.

`1` Instruction caches enabled at EL2. If HSCTLR.M is set to 0, instruction accesses from stage 1 of the EL2 translation regime are to Normal memory, Outer Shareable, Inner Write-Through, Outer Write-Through.

When this bit is 0, all EL2 Normal memory instruction accesses are Non-cacheable.

The reset value for this field is UNKNOWN.

C, [2]

Cache enable. This is an enable bit for data and unified caches at EL2:

`0` Data and unified caches disabled at EL2. This is the reset value.

`1` Data and unified caches enabled at EL2.

When this bit is 0, all EL2 Normal memory data accesses and all accesses to the EL2 translation tables are Non-cacheable.

The reset value for this field is UNKNOWN.

M, [0]

MMU enable. This is a global enable bit for the EL2 stage 1 MMU:

`0` EL2 stage 1 MMU disabled. This is the reset value.

`1` EL2 stage 1 MMU enabled.

The reset value for this field is UNKNOWN.

Configurations

HSCTLR is architecturally mapped to AArch64 register SCTLR_EL2. See *B2.90 SCTLR_EL2, System Control Register, EL2* on page B2-418.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

B1.55 HSR, Hyp Syndrome Register

The HSR holds syndrome information for an exception taken to Hyp mode.

Bit field descriptions

HSR is a 32-bit register, and is part of:

- The Virtualization registers functional group.
- The Exception and fault handling registers functional group.

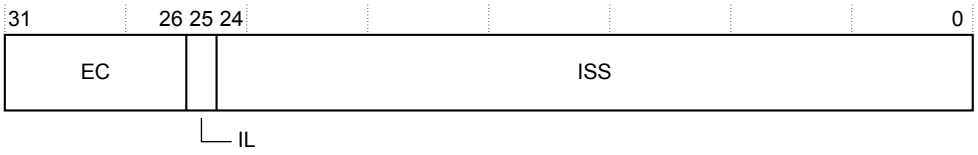

Figure B1-39 HSR bit assignments

EC, [31:26]
Exception class. The exception class for the exception that is taken in Hyp mode. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile* for more information.

IL, [25]
Instruction length. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile* for more information.

ISS, [24:0]
Instruction specific syndrome. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile* for more information. The interpretation of this field depends on the value of the EC field. See *B1.55.1 Encoding of ISS[24:20] when HSR[31:30] is 0b00* on page B1-208.

Configurations
HSR is architecturally mapped to AArch64 register ESR_EL2. See *B2.48 ESR_EL2, Exception Syndrome Register, EL2* on page B2-353.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

This section contains the following subsection:

B1.55.1 Encoding of ISS[24:20] when HSR[31:30] is 0b00

For EC values that are non-zero and have the two most-significant bits 0b00, ISS[24:20] provides the condition field for the trapped instruction, together with a valid flag for this field.

The encoding of this part of the ISS field is:

CV, ISS[24]
Condition valid. Possible values of this bit are:

`0` The COND field is not valid.

`1` The COND field is valid.

When an instruction is trapped, CV is set to 1.

COND, ISS[23:20]
The Condition field for the trapped instruction. This field is valid only when CV is set to 1.

If CV is set to 0, this field is RES0.

When an instruction is trapped, the COND field is set to the condition the instruction was executed with.

When reporting an SEI, the following occurs:

- AET always reports an uncontainable error (UC) with value `0b00`.
- EA is RES0.
- DFSC is always at `0b010001`.

When reporting a synchronous external data abort, EA is RES0.

When reporting a synchronous external prefetch abort, EA is RES0.

B1.56 HTTBR, Hyp Translation Table Base Register

The HTTBR holds the base address of the translation table for the stage 1 translation of memory accesses from Hyp mode.

Bit field descriptions

HTTBR is a 64-bit register.

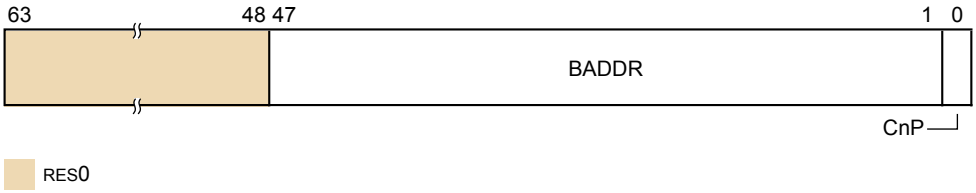

Figure B1-40 HTTBR bit assignments

CnP, [0]

Common not Private. The possible values are:

0 CnP is not supported.

1 CnP is supported.

Configurations

AArch32 System register HTTBR is architecturally mapped to AArch64 System register TTBR0_EL2.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

B1.57 ID_AFR0, Auxiliary Feature Register 0

The ID_AFR0 provides information about the IMPLEMENTATION DEFINED features of the PE in AArch32. This register is not used in the Cortex-A75 core.

Bit field descriptions

ID_AFR0 is a 32-bit register, and is part of the Identification registers functional group.

This register is Read Only.

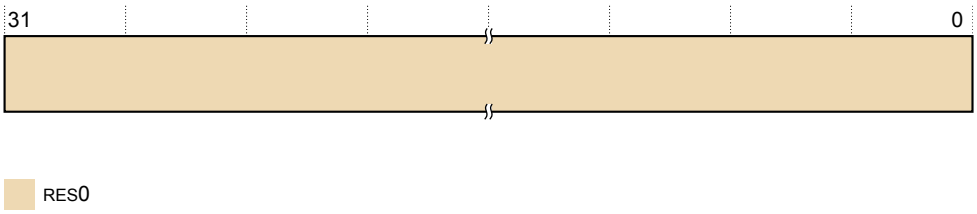

Figure B1-41 ID_AFR0 bit assignments

RES0, [31:0]

Reserved, RES0.

Configurations

AArch32 System register ID_AFR0 is architecturally mapped to AArch64 System register ID_AFR0_EL1. See *B2.62 ID_AFR0_EL1, AArch32 Auxiliary Feature Register 0, EL1* on page B2-373.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

B1.58 ID_DFR0, Debug Feature Register 0

The ID_DFR0 provides top-level information about the debug system in AArch32.

Bit field descriptions

ID_DFR0 is a 32-bit register, and is part of the Identification registers functional group.

This register is Read Only.

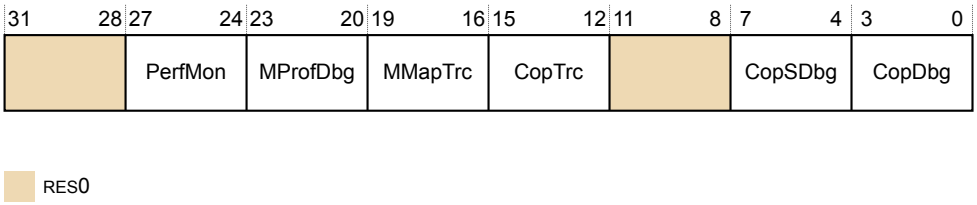

Figure B1-42 ID_DFR0 bit assignments

RES0, [31:28]

RES0 Reserved.

PerfMon, [27:24]

Indicates support for performance monitor model:

`0x4` Support for *Performance Monitor Unit version 3* (PMUv3) system registers, with a 16-bit evtCount field.

MProfDbg, [23:20]

Indicates support for memory-mapped debug model for M profile cores:

`0x0` This product does not support M profile Debug architecture.

MMapTrc, [19:16]

Indicates support for memory-mapped trace model:

`0x1` Support for Arm trace architecture, with memory-mapped access.

In the Trace registers, the ETMIDR gives more information about the implementation.

CopTrc, [15:12]

Indicates support for coprocessor-based trace model:

`0x0` This product does not support Arm trace architecture.

RES0, [11:8]

RES0 Reserved.

CopSDbg, [7:4]

Indicates support for coprocessor-based Secure debug model:

`0x8` This product supports v8.2 Debug architecture.

CopDbg, [3:0]

Indicates support for coprocessor-based debug model:

`0x8` This product supports v8.2 Debug architecture.

Configurations

ID_DFR0 is architecturally mapped to AArch64 register ID_DFR0_EL1. See *B2.63 ID_DFR0_EL1, AArch32 Debug Feature Register 0, EL1* on page B2-374.

There is one copy of this register that is used in both Secure and Non-secure states.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

B1.59 ID_ISAR0, Instruction Set Attribute Register 0

The ID_ISAR0 provides information about the instruction sets implemented by the core in AArch32.

Bit field descriptions

ID_ISAR0 is a 32-bit register, and is part of the Identification registers functional group.

This register is Read Only.

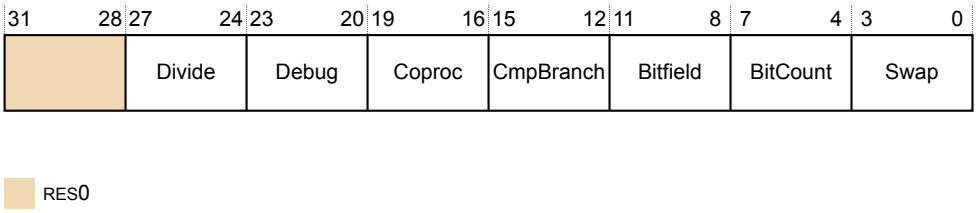

Figure B1-43 ID_ISAR0 bit assignments

RES0, [31:28]

RES0 Reserved.

Divide, [27:24]

Indicates the implemented Divide instructions:

0x2
- SDIV and UDIV in the T32 instruction set.
- SDIV and UDIV in the A32 instruction set.

Debug, [23:20]

Indicates the implemented Debug instructions:

0x1 BKPT.

Coproc, [19:16]

Indicates the implemented Coprocessor instructions:

0x0 None implemented, except for instructions separately attributed by the architecture to provide access to AArch32 System registers and System instructions.

CmpBranch, [15:12]

Indicates the implemented combined Compare and Branch instructions in the T32 instruction set:

0x1 CBNZ and CBZ.

Bitfield, [11:8]

Indicates the implemented bit field instructions:

0x1 BFC, BFI, SBFX, and UBFX.

BitCount, [7:4]

Indicates the implemented Bit Counting instructions:

0x1 CLZ.

Swap, [3:0]

Indicates the implemented Swap instructions in the A32 instruction set:

0x0 None implemented.

Configurations

ID_ISAR0 is architecturally mapped to AArch64 register ID_ISAR0_EL1. See *B2.64 ID_ISAR0_EL1, AArch32 Instruction Set Attribute Register 0, EL1* on page B2-376.

There is one copy of this register that is used in both Secure and Non-secure states.

Must be interpreted with ID_ISAR1, ID_ISAR2, ID_ISAR3, ID_ISAR4, and ID_ISAR5. See:

- *B1.60 ID_ISAR1, Instruction Set Attribute Register 1* on page B1-216.
- *B1.61 ID_ISAR2, Instruction Set Attribute Register 2* on page B1-218.
- *B1.62 ID_ISAR3, Instruction Set Attribute Register 3* on page B1-220.
- *B1.63 ID_ISAR4, Instruction Set Attribute Register 4* on page B1-222.
- *B1.64 ID_ISAR5, Instruction Set Attribute Register 5* on page B1-224.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

B1.60 ID_ISAR1, Instruction Set Attribute Register 1

The ID_ISAR1 provides information about the instruction sets implemented by the core in AArch32.

Bit field descriptions

ID_ISAR1 is a 32-bit register, and is part of the Identification registers functional group.

This register is Read Only.

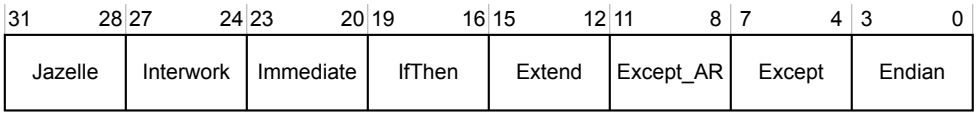

Figure B1-44 ID_ISAR1 bit assignments

Jazelle, [31:28]
Indicates the implemented Jazelle Extension instructions:

`0x1` The `BXJ` instruction, and the J bit in the PSR.

Interwork, [27:24]
Indicates the implemented Interworking instructions:

`0x3`
- The `BX` instruction, and the T bit in the PSR.
- The `BLX` instruction. PC loads have `BX`-like behavior.
- Data-processing instructions in the A32 instruction set with the PC as the destination and the S bit clear have `BX`-like behavior.

Immediate, [23:20]
Indicates the implemented data-processing instructions with long immediates:

`0x1`
- The `MOVT` instruction.
- The `MOV` instruction encodings with zero-extended 16-bit immediates.
- The T32 `ADD` and `SUB` instruction encodings with zero-extended 12-bit immediates, and other `ADD`, `ADR`, and `SUB` encodings cross-referenced by the pseudocode for those encodings.

IfThen, [19:16]
Indicates the implemented If-Then instructions in the T32 instruction set:

`0x1` The `IT` instructions, and the IT bits in the PSRs.

Extend, [15:12]
Indicates the implemented Extend instructions:

`0x2`
- The `SXTB`, `SXTH`, `UXTB`, and `UXTH` instructions.
- The `SXTB16`, `SXTAB`, `SXTAB16`, `SXTAH`, `UXTB16`, `UXTAB`, `UXTAB16`, and `UXTAH` instructions.

Except_AR, [11:8]
Indicates the implemented A profile exception-handling instructions:

`0x1` The `SRS` and `RFE` instructions, and the A and R profile forms of the `CPS` instruction.

Except, [7:4]
Indicates the implemented exception-handling instructions in the A32 instruction set:

`0x1` The `LDM` (exception return), `LDM` (user registers), and `STM` (user registers) instruction versions.

Endian, [3:0]
Indicates the implemented Endian instructions:

`0x1` The `SETEND` instruction, and the E bit in the PSRs.

Configurations

ID_ISAR1 is architecturally mapped to AArch64 register ID_ISAR1_EL1. See *B2.65 ID_ISAR1_EL1, AArch32 Instruction Set Attribute Register 1, EL1* on page B2-378.

There is one copy of this register that is used in both Secure and Non-secure states.

Must be interpreted with ID_ISAR0, ID_ISAR2, ID_ISAR3, ID_ISAR4 and ID_ISAR5. See:

- *B1.59 ID_ISAR0, Instruction Set Attribute Register 0* on page B1-214.
- *B1.61 ID_ISAR2, Instruction Set Attribute Register 2* on page B1-218.
- *B1.62 ID_ISAR3, Instruction Set Attribute Register 3* on page B1-220.
- *B1.63 ID_ISAR4, Instruction Set Attribute Register 4* on page B1-222.
- *B1.64 ID_ISAR5, Instruction Set Attribute Register 5* on page B1-224.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

B1.61 ID_ISAR2, Instruction Set Attribute Register 2

The ID_ISAR2 provides information about the instruction sets implemented by the core in AArch32.

Bit field descriptions

ID_ISAR2 is a 32-bit register, and is part of the Identification registers functional group.

This register is Read Only.

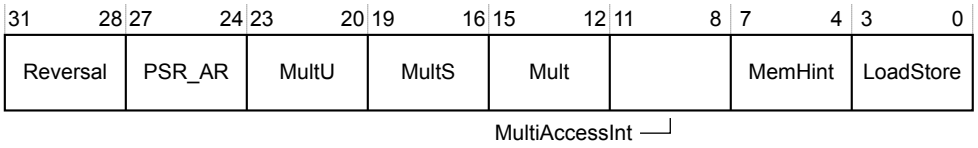

Figure B1-45 ID_ISAR2 bit assignments

Reversal, [31:28]

Indicates the implemented Reversal instructions:

`0x2` • The `REV`, `REV16`, and `REVSH` instructions.
• The `RBIT` instruction.

PSR_AR, [27:24]

Indicates the implemented A and R profile instructions to manipulate the PSR:

`0x1` The `MRS` and `MSR` instructions, and the exception return forms of data-processing instructions.

The exception return forms of the data-processing instructions are:

- In the A32 instruction set, data-processing instructions with the PC as the destination and the S bit set.
- In the T32 instruction set, the `SUBS PC, LR, #N` instruction.

MultU, [23:20]

Indicates the implemented advanced unsigned Multiply instructions:

`0x2` • The `UMULL` and `UMLAL` instructions.
• The `UMAAL` instruction.

MultS, [19:16]

Indicates the implemented advanced signed Multiply instructions.

`0x3` • The `SMULL` and `SMLAL` instructions.
• The `SMLABB`, `SMLABT`, `SMLALBB`, `SMLALBT`, `SMLALTB`, `SMLALTT`, `SMLATB`, `SMLATT`, `SMLAWB`, `SMLAWT`, `SMULBB`, `SMULBT`, `SMULTB`, `SMULTT`, `SMULWB`, `SMULWT` instructions, and the Q bit in the PSRs.
• The `SMLAD`, `SMLADX`, `SMLALD`, `SMLALDX`, `SMLSD`, `SMLSDX`, `SMLSLD`, `SMLSLDX`, `SMMLA`, `SMMLAR`, `SMMLS`, `SMMLSR`, `SMMUL`, `SMMULR`, `SMUAD`, `SMUADX`, `SMUSD`, and `SMUSDX` instructions.

Mult, [15:12]

Indicates the implemented additional Multiply instructions:

`0x2` • The `MUL` instruction.
• The `MLA` instruction.
• The `MLS` instruction.

MultiAccessInt, [11:8]

Indicates the support for interruptible multi-access instructions:

`0x0` No support. This means that the LDM and STM instructions are not interruptible.

MemHint, [7:4]

Indicates the implemented memory hint instructions:

`0x4`
- The `PLD` instruction.
- The `PLI` instruction.
- The `PLDW` instruction.

LoadStore, [3:0]

Indicates the implemented additional load/store instructions:

`0x2`
- The `LDRD` and `STRD` instructions.
- The Load Acquire (`LDAB`, `LDAH`, `LDA`, `LDAEXB`, `LDAEXH`, `LDAEX`, and `LDAEXD`) and Store Release (`STLB`, `STLH`, `STL`, `STLEXB`, `STLEXH`, `STLEX`, and `STLEXD`) instructions.

Configurations

ID_ISAR2 is architecturally mapped to AArch64 register ID_ISAR2_EL1. See *B2.66 ID_ISAR2_EL1, AArch32 Instruction Set Attribute Register 2, EL1* on page B2-380.

There is one copy of this register that is used in both Secure and Non-secure states.

Must be interpreted with ID_ISAR0, ID_ISAR1, ID_ISAR3, ID_ISAR4 and ID_ISAR5. See:
- *B1.59 ID_ISAR0, Instruction Set Attribute Register 0* on page B1-214.
- *B1.60 ID_ISAR1, Instruction Set Attribute Register 1* on page B1-216.
- *B1.62 ID_ISAR3, Instruction Set Attribute Register 3* on page B1-220.
- *B1.63 ID_ISAR4, Instruction Set Attribute Register 4* on page B1-222.
- *B1.64 ID_ISAR5, Instruction Set Attribute Register 5* on page B1-224.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

B1.62 ID_ISAR3, Instruction Set Attribute Register 3

The ID_ISAR3 provides information about the instruction sets implemented by the core in AArch32.

Bit field descriptions

ID_ISAR3 is a 32-bit register, and is part of the Identification registers functional group.

This register is Read Only.

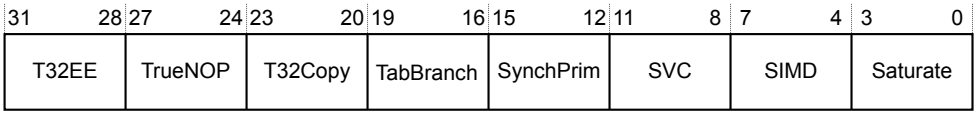

Figure B1-46 ID_ISAR3 bit assignments

T32EE, [31:28]
Indicates the implemented Thumb Execution Environment (T32EE) instructions:

`0x0` None implemented.

TrueNOP, [27:24]
Indicates the implemented true NOP instructions:

`0x1` True `NOP` instructions in both the A32 and T32 instruction sets, and additional NOP-compatible hints.

T32Copy, [23:20]
Indicates the support for T32 non flag-setting `MOV` instructions:

`0x1` Support for T32 instruction set encoding T1 of the `MOV` (register) instruction, copying from a low register to a low register.

TabBranch, [19:16]
Indicates the implemented Table Branch instructions in the T32 instruction set.

`0x1` The `TBB` and `TBH` instructions.

SynchPrim, [15:12]
Used in conjunction with ID_ISAR4.SynchPrim_frac to indicate the implemented Synchronization Primitive instructions.

`0x2`
- The `LDREX` and `STREX` instructions.
- The `CLREX`, `LDREXB`, `STREXB`, and `STREXH` instructions.
- The `LDREXD` and `STREXD` instructions.

SVC, [11:8]
Indicates the implemented SVC instructions:

`0x1` The `SVC` instruction.

SIMD, [7:4]
Indicates the implemented *Single Instruction Multiple Data* (SIMD) instructions.

`0x3`
- The `SSAT` and `USAT` instructions, and the Q bit in the PSRs.
- The `PKHBT`, `PKHTB`, `QADD16`, `QADD8`, `QASX`, `QSUB16`, `QSUB8`, `QSAX`, `SADD16`, `SADD8`, `SASX`, `SEL`, `SHADD16`, `SHADD8`, `SHASX`, `SHSUB16`, `SHSUB8`, `SHSAX`, `SSAT16`, `SSUB16`, `SSUB8`, `SSAX`, `SXTAB16`, `SXTB16`, `UADD16`, `UADD8`, `UASX`, `UHADD16`, `UHADD8`, `UHASX`, `UHSUB16`, `UHSUB8`, `UHSAX`, `UQADD16`, `UQADD8`, `UQASX`, `UQSUB16`, `UQSUB8`, `UQSAX`, `USAD8`, `USADA8`, `USAT16`, `USUB16`, `USUB8`, `USAX`, `UXTAB16`, `UXTB16` instructions, and the GE[3:0] bits in the PSRs.

The SIMD field relates only to implemented instructions that perform SIMD operations on the general-purpose registers. In an implementation that supports Advanced SIMD and floating-point instructions, MVFR0, MVFR1, and MVFR2 give information about the implemented Advanced SIMD instructions.

Saturate, [3:0]

Indicates the implemented Saturate instructions:

`0x1` The `QADD`, `QDADD`, `QDSUB`, `QSUB` and the Q bit in the PSRs.

Configurations

ID_ISAR3 is architecturally mapped to AArch64 register ID_ISAR3_EL1. See *B2.67 ID_ISAR3_EL1, AArch32 Instruction Set Attribute Register 3, EL1* on page B2-382.

There is one copy of this register that is used in both Secure and Non-secure states.

Must be interpreted with ID_ISAR0, ID_ISAR1, ID_ISAR2, ID_ISAR4, and ID_ISAR5. See:
- *B1.59 ID_ISAR0, Instruction Set Attribute Register 0* on page B1-214.
- *B1.60 ID_ISAR1, Instruction Set Attribute Register 1* on page B1-216.
- *B1.61 ID_ISAR2, Instruction Set Attribute Register 2* on page B1-218.
- *B1.63 ID_ISAR4, Instruction Set Attribute Register 4* on page B1-222.
- *B1.64 ID_ISAR5, Instruction Set Attribute Register 5* on page B1-224.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

B1.63 ID_ISAR4, Instruction Set Attribute Register 4

The ID_ISAR4 provides information about the instruction sets implemented by the core in AArch32.

Bit field descriptions

ID_ISAR4 is a 32-bit register, and is part of the Identification registers functional group.

This register is Read Only.

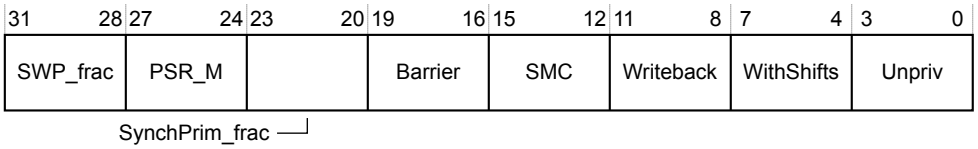

Figure B1-47 ID_ISAR4 bit assignments

SWP_frac, [31:28]
Indicates support for the memory system locking the bus for `SWP` or `SWPB` instructions:

`0x0` `SWP` and `SWPB` instructions not implemented.

PSR_M, [27:24]
Indicates the implemented M profile instructions to modify the PSRs:

`0x0` None implemented.

SynchPrim_frac, [23:20]
This field is used with the ID_ISAR3.SynchPrim field to indicate the implemented Synchronization Primitive instructions:

`0x0`
- The `LDREX` and `STREX` instructions.
- The `CLREX`, `LDREXB`, `LDREXH`, `STREXB`, and `STREXH` instructions.
- The `LDREXD` and `STREXD` instructions.

Barrier, [19:16]
Indicates the supported Barrier instructions in the A32 and T32 instruction sets:

`0x1` The `DMB`, `DSB`, and `ISB` barrier instructions.

SMC, [15:12]
Indicates the implemented `SMC` instructions:

`0x1` The `SMC` instruction.

Write-Back, [11:8]
Indicates the support for Write-Back addressing modes:

`0x1` Core supports all the Write-Back addressing modes defined in Armv8.

WithShifts, [7:4]
Indicates the support for instructions with shifts:

`0x4`
- Support for shifts of loads and stores over the range LSL 0-3.
- Support for other constant shift options, both on load/store and other instructions.
- Support for register-controlled shift options.

Unpriv, [3:0]
Indicates the implemented unprivileged instructions:

`0x2`
- The `LDRBT`, `LDRT`, `STRBT`, and `STRT` instructions.
- The `LDRHT`, `LDRSBT`, `LDRSHT`, and `STRHT` instructions.

Configurations

ID_ISAR4 is architecturally mapped to AArch64 register ID_ISAR4_EL1. See *B2.68 ID_ISAR4_EL1, AArch32 Instruction Set Attribute Register 4, EL1* on page B2-384.

There is one copy of this register that is used in both Secure and Non-secure states.

Must be interpreted with ID_ISAR0, ID_ISAR1, ID_ISAR2, ID_ISAR3, and ID_ISAR5. See:

- *B1.59 ID_ISAR0, Instruction Set Attribute Register 0* on page B1-214.
- *B1.60 ID_ISAR1, Instruction Set Attribute Register 1* on page B1-216.
- *B1.61 ID_ISAR2, Instruction Set Attribute Register 2* on page B1-218.
- *B1.62 ID_ISAR3, Instruction Set Attribute Register 3* on page B1-220.
- *B1.64 ID_ISAR5, Instruction Set Attribute Register 5* on page B1-224.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

B1.64 ID_ISAR5, Instruction Set Attribute Register 5

The ID_ISAR5 provides information about the instruction sets that the core implements.

Bit field descriptions

ID_ISAR5 is a 32-bit register, and is part of the Identification registers functional group.

This register is Read Only.

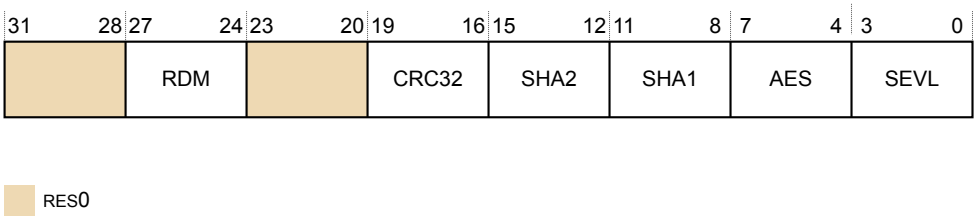

Figure B1-48 ID_ISAR5 bit assignments

RES0, [31:28]

RES0 Reserved.

RDM, [27:24]

VQRDMLAH and VQRDMLSH instructions in AArch32. The value is:

0x1 VQRDMLAH and VQRDMLSH instructions are implemented.

RES0, [23:20]

RES0 Reserved.

CRC32, [19:16]

Indicates whether CRC32 instructions are implemented in AArch32 state. The value is:

0x1 CRC32B, CRC32H, CRC32W, CRC32CB, CRC32CH, and CRC32CW instructions are implemented.

SHA2, [15:12]

Indicates whether SHA2 instructions are implemented in AArch32 state:

0x0 No SHA2 instructions implemented. This is the value when the Cryptographic Extensions are not implemented or are disabled.

0x1 SHA256H, SHA256H2, SHA256SU0, and SHA256SU1 implemented. This is the value when the Cryptographic Extensions are implemented and enabled.

SHA1, [11:8]

Indicates whether SHA1 instructions are implemented in AArch32 state. Defined values are:

0x0 No SHA1 instructions implemented. This is the value when the Cryptographic Extensions are not implemented or are disabled.

0x1 SHA1C, SHA1P, SHA1M, SHA1H, SHA1SU0, and SHA1SU1 implemented. This is the value when the Cryptographic Extensions are implemented and enabled.

AES, [7:4]

Indicates whether AES instructions are implemented in AArch32 state. Defined values are:

0x0 No AES instructions implemented. This is the value when the Cryptographic Extensions are not implemented or are disabled.

`0x2` • AESE, AESD, AESMC, and AESIMC implemented.
• PMULL/PMULL2 instructions operating on 64-bit data quantities.

This is the value when the Cryptographic Extensions are implemented and enabled.

SEVL, [3:0]

Indicates whether the `SEVL` instruction is implemented in AArch32. The value is:

`0x1` `SEVL` is implemented as send event local.

Configurations

ID_ISAR5 is architecturally mapped to AArch64 register ID_ISAR5_EL1. See *B2.69 ID_ISAR5_EL1, AArch32 Instruction Set Attribute Register 5, EL1* on page B2-386.

There is one copy of this register that is used in both Secure and Non-secure states.

ID_ISAR5 must be interpreted with ID_ISAR0, ID_ISAR1, ID_ISAR2, ID_ISAR3, and ID_ISAR4. See:

- *B1.59 ID_ISAR0, Instruction Set Attribute Register 0* on page B1-214.
- *B1.60 ID_ISAR1, Instruction Set Attribute Register 1* on page B1-216.
- *B1.61 ID_ISAR2, Instruction Set Attribute Register 2* on page B1-218.
- *B1.62 ID_ISAR3, Instruction Set Attribute Register 3* on page B1-220.
- *B1.63 ID_ISAR4, Instruction Set Attribute Register 4* on page B1-222.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

B1.65 ID_ISAR6, Instruction Set Attribute Register 6

The ID_ISAR6 provides information about the instruction sets that the core implements.

Bit field descriptions

ID_ISAR6 is a 32-bit register, and is part of the Identification registers functional group.

This register is Read Only.

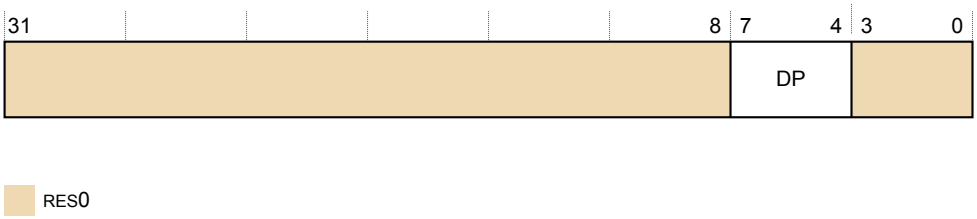

Figure B1-49 ID_ISAR6 bit assignments

RES0, [31:8]

RES0 Reserved.

DP, [7:4]

UDOT and SDOT instructions. The value is:

0b0001 UDOT and SDOT instructions are implemented.

RES0, [3:0]

RES0 Reserved.

Configurations

ID_ISAR6 is architecturally mapped to AArch64 register ID_ISAR6_EL1. See *B2.70 ID_ISAR6_EL1, AArch32 Instruction Set Attribute Register 6, EL1* on page B2-388.

There is one copy of this register that is used in both Secure and Non-secure states.

ID_ISAR6 must be interpreted with ID_ISAR0, ID_ISAR1, ID_ISAR2, ID_ISAR3, ID_ISAR4, and ID_ISAR5. See:

- *B1.59 ID_ISAR0, Instruction Set Attribute Register 0* on page B1-214.
- *B1.60 ID_ISAR1, Instruction Set Attribute Register 1* on page B1-216.
- *B1.61 ID_ISAR2, Instruction Set Attribute Register 2* on page B1-218.
- *B1.62 ID_ISAR3, Instruction Set Attribute Register 3* on page B1-220.
- *B1.63 ID_ISAR4, Instruction Set Attribute Register 4* on page B1-222.
- *B1.64 ID_ISAR5, Instruction Set Attribute Register 5* on page B1-224.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

B1.66 ID_MMFR0, Memory Model Feature Register 0

The ID_MMFR0 provides information about the memory model and memory management support in AArch32.

Bit field descriptions

ID_MMFR0 is a 32-bit register, and is part of the Identification registers functional group.

This register is Read Only.

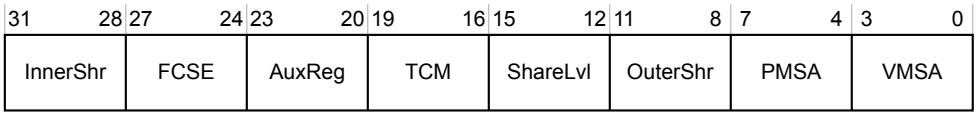

Figure B1-50 ID_MMFR0 bit assignments

InnerShr, [31:28]
Indicates the innermost shareability domain implemented:

`0x1` Implemented with hardware coherency support.

FCSE, [27:24]
Indicates support for *Fast Context Switch Extension* (FCSE):

`0x0` Not supported.

AuxReg, [23:20]
Indicates support for Auxiliary registers:

`0x2` Support for Auxiliary Fault Status Registers (AIFSR and ADFSR) and Auxiliary Control Register.

TCM, [19:16]
Indicates support for TCMs and associated DMAs:

`0x0` Not supported.

ShareLvl, [15:12]
Indicates the number of shareability levels implemented:

`0x1` Two levels of shareability implemented.

OuterShr, [11:8]
Indicates the outermost shareability domain implemented:

`0x1` Implemented with hardware coherency support.

PMSA, [7:4]
Indicates support for a *Protected Memory System Architecture* (PMSA):

`0x0` Not supported.

VMSA, [3:0]
Indicates support for a *Virtual Memory System Architecture* (VMSA).

`0x5` Support for:
- VMSAv7, with support for remapping and the Access flag.
- The PXN bit in the Short-descriptor translation table format descriptors.
- The Long-descriptor translation table format.

Configurations

ID_MMFR0 is architecturally mapped to AArch64 register ID_MMFR0_EL1. See *B2.71 ID_MMFR0_EL1, AArch32 Memory Model Feature Register 0, EL1* on page B2-389.

There is one copy of this register that is used in both Secure and Non-secure states.

Must be interpreted with ID_MMFR1, ID_MMFR2, ID_MMFR3, and ID_MMFR4. See:

- *B1.67 ID_MMFR1, Memory Model Feature Register 1* on page B1-229.
- *B1.68 ID_MMFR2, Memory Model Feature Register 2* on page B1-231.
- *B1.69 ID_MMFR3, Memory Model Feature Register 3* on page B1-233.
- *B1.70 ID_MMFR4, Memory Model Feature Register 4* on page B1-235.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

B1.67 ID_MMFR1, Memory Model Feature Register 1

The ID_MMFR1 provides information about the memory model and memory management support in AArch32.

Bit field descriptions

ID_MMFR1 is a 32-bit register, and is part of the Identification registers functional group.

This register is Read Only.

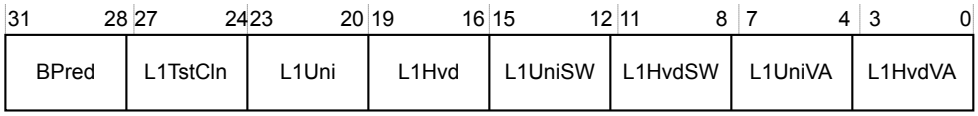

Figure B1-51 ID_MMFR1 bit assignments

BPred, [31:28]
Indicates branch predictor management requirements:

`0x4` For execution correctness, branch predictor requires no flushing at any time.

L1TstCln, [27:24]
Indicates the supported L1 Data cache test and clean operations, for Harvard or unified cache implementation:

`0x0` None supported.

L1Uni, [23:20]
Indicates the supported entire L1 cache maintenance operations, for a unified cache implementation:

`0x0` None supported.

L1Hvd, [19:16]
Indicates the supported entire L1 cache maintenance operations, for a Harvard cache implementation:

`0x0` None supported.

L1UniSW, [15:12]
Indicates the supported L1 cache line maintenance operations by set/way, for a unified cache implementation:

`0x0` None supported.

L1HvdSW, [11:8]
Indicates the supported L1 cache line maintenance operations by set/way, for a Harvard cache implementation:

`0x0` None supported.

L1UniVA, [7:4]
Indicates the supported L1 cache line maintenance operations by VA, for a unified cache implementation:

`0x0` None supported.

L1HvdVA, [3:0]
Indicates the supported L1 cache line maintenance operations by VA, for a Harvard cache implementation:

`0x0` None supported.

Configurations

ID_MMFR1 is architecturally mapped to AArch64 register ID_MMFR1_EL1. See *B2.72 ID_MMFR1_EL1, AArch32 Memory Model Feature Register 1, EL1* on page B2-391.

There is one copy of this register that is used in both Secure and Non-secure states.

Must be interpreted with ID_MMFR0, ID_MMFR2, ID_MMFR3, and ID_MMFR4. See:

- *B1.66 ID_MMFR0, Memory Model Feature Register 0* on page B1-227.
- *B1.68 ID_MMFR2, Memory Model Feature Register 2* on page B1-231.
- *B1.69 ID_MMFR3, Memory Model Feature Register 3* on page B1-233.
- *B1.70 ID_MMFR4, Memory Model Feature Register 4* on page B1-235.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

B1.68 ID_MMFR2, Memory Model Feature Register 2

The ID_MMFR2 provides information about the implemented memory model and memory management support in AArch32.

Bit field descriptions

ID_MMFR2 is a 32-bit register, and is part of the Identification registers functional group.

This register is Read Only.

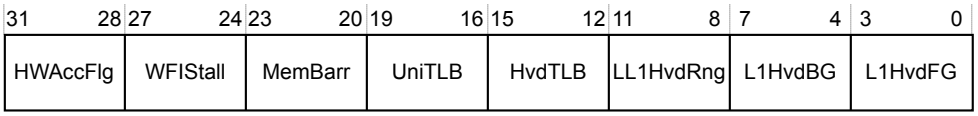

Figure B1-52 ID_MMFR2 bit assignments

HWAccFlg, [31:28]
Hardware Access Flag. Indicates support for a Hardware Access flag, as part of the VMSAv7 implementation:

`0x0` Not supported.

WFIStall, [27:24]
Wait For Interrupt Stall. Indicates the support for *Wait For Interrupt* (WFI) stalling:

`0x1` Support for WFI stalling.

MemBarr, [23:20]
Memory Barrier. Indicates the supported CP15 memory barrier operations.

`0x2` Supported CP15 memory barrier operations are:
- *Data Synchronization Barrier* (DSB).
- *Instruction Synchronization Barrier* (ISB).
- *Data Memory Barrier* (DMB).

UniTLB, [19:16]
Unified TLB. Indicates the supported TLB maintenance operations, for a unified TLB implementation.

`0x6` Supported unified TLB maintenance operations are:
- Invalidate all entries in the TLB.
- Invalidate TLB entry by VA.
- Invalidate TLB entries by ASID match.
- Invalidate instruction TLB and data TLB entries by VA All ASID. This is a shared unified TLB operation.
- Invalidate Hyp mode unified TLB entry by VA.
- Invalidate entire Non-secure PL1 and PL0 unified TLB.
- Invalidate entire Hyp mode unified TLB.
- `TLBIMVALIS`, `TLBIMVAALIS`, `TLBIMVALHIS`, `TLBIMVAL`, `TLBIMVAAL`, and `TLBIMVALH`.
- `TLBIIPAS2IS`, `TLBIIPAS2LIS`, `TLBIIPAS2`, and `TLBIIPAS2L`.

HvdTLB, [15:12]
Harvard TLB. Indicates the supported TLB maintenance operations, for a Harvard TLB implementation:

`0x0` Not supported.

LL1HvdRng, [11:8]
L1 Harvard cache Range. Indicates the supported L1 cache maintenance range operations, for a Harvard cache implementation:

`0x0` Not supported.

L1HvdBG, [7:4]

L1 Harvard cache Background fetch. Indicates the supported L1 cache background prefetch operations, for a Harvard cache implementation:

`0x0` Not supported.

L1HvdFG, [3:0]

L1 Harvard cache Foreground fetch. Indicates the supported L1 cache foreground prefetch operations, for a Harvard cache implementation:

`0x0` Not supported.

Configurations

ID_MMFR2 is architecturally mapped to AArch64 register ID_MMFR2_EL1. See *B2.73 ID_MMFR2_EL1, AArch32 Memory Model Feature Register 2, EL1* on page B2-393.

There is one copy of this register that is used in both Secure and Non-secure states.

Must be interpreted with ID_MMFR0, ID_MMFR1, ID_MMFR3, and ID_MMFR4. See:

- *B1.66 ID_MMFR0, Memory Model Feature Register 0* on page B1-227.
- *B1.67 ID_MMFR1, Memory Model Feature Register 1* on page B1-229.
- *B1.69 ID_MMFR3, Memory Model Feature Register 3* on page B1-233.
- *B1.70 ID_MMFR4, Memory Model Feature Register 4* on page B1-235.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

B1.69 ID_MMFR3, Memory Model Feature Register 3

The ID_MMFR3 provides information about the memory model and memory management support in AArch32.

Bit field descriptions

ID_MMFR3 is a 32-bit register, and is part of the Identification registers functional group.

This register is Read Only.

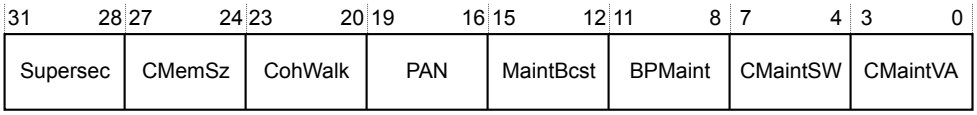

Figure B1-53 ID_MMFR3 bit assignments

Supersec, [31:28]
Supersections. Indicates support for supersections:

`0x0` Supersections supported.

CMemSz, [27:24]
Cached Memory Size. Indicates the size of physical memory supported by the core caches:

`0x2` 1TByte, corresponding to a 40-bit physical address range.

CohWalk, [23:20]
Coherent walk. Indicates whether translation table updates require a clean to the point of unification:

`0x1` Updates to the translation tables do not require a clean to the point of unification to ensure visibility by subsequent translation table walks.

PAN, [19:16]
Privileged Access Never. Indicates support for the PAN bit in CPSR, SPSR, and DSPSR in AArch32.

`0x2`
- PAN supported.
- ATS1CPRP and ATS1CPWP supported.

MaintBcst, [15:12]
Maintenance broadcast. Indicates whether cache, TLB, and branch predictor operations are broadcast:

`0x2` Cache, TLB, and branch predictor operations affect structures according to shareability and defined behavior of instructions.

BPMaint, [11:8]
Branch predictor maintenance. Indicates the supported branch predictor maintenance operations.

`0x2` Supported branch predictor maintenance operations are:
- Invalidate all branch predictors.
- Invalidate branch predictors by *Virtual Address* (VA).

CMaintSW, [7:4]
Cache maintenance by set/way. Indicates the supported cache maintenance operations by set/way.

`0x1` Supported hierarchical cache maintenance operations by set/way are:
- Invalidate data cache by set/way.
- Clean data cache by set/way.
- Clean and invalidate data cache by set/way.

CMaintVA, [3:0]

Cache maintenance by VA. Indicates the supported cache maintenance operations by VA.

`0x1` Supported hierarchical cache maintenance operations by VA are:
- Invalidate data cache by VA.
- Clean data cache by VA.
- Clean and invalidate data cache by VA.
- Invalidate instruction cache by VA.
- Invalidate all instruction cache entries.

Configurations

ID_MMFR3 is architecturally mapped to AArch64 register ID_MMFR3_EL1. See *B2.74 ID_MMFR3_EL1, AArch32 Memory Model Feature Register 3, EL1* on page B2-395.

There is one copy of this register that is used in both Secure and Non-secure states.

Must be interpreted with ID_MMFR0, ID_MMFR1, ID_MMFR2, and ID_MMFR4. See:
- *B1.66 ID_MMFR0, Memory Model Feature Register 0* on page B1-227.
- *B1.67 ID_MMFR1, Memory Model Feature Register 1* on page B1-229.
- *B1.68 ID_MMFR2, Memory Model Feature Register 2* on page B1-231.
- *B1.70 ID_MMFR4, Memory Model Feature Register 4* on page B1-235.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

B1.70 ID_MMFR4, Memory Model Feature Register 4

The ID_MMFR4 provides information about the memory model and memory management support in AArch32.

Bit field descriptions

ID_MMFR4 is a 32-bit register, and is part of the Identification registers functional group.

This register is Read Only.

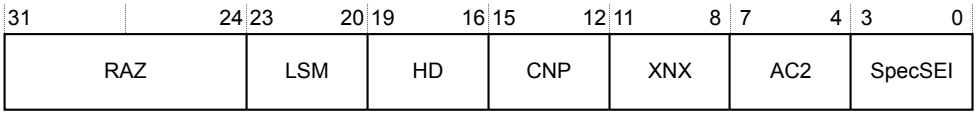

Figure B1-54 ID_MMFR4 bit assignments

RAZ, [31:24]

Read-as-zero.

LSM, [23:20]

Load/Store Multiple. Indicates whether adjacent loads or stores can be combined. The value is:

`0x0` LSMAOE and nTLSMD bit not supported.

HD, [19:16]

Hierarchical Disables. Enables an operating system or hypervisor to hand over up to 4 bits of the last level page table descriptor (bits[62:59] of the page table entry) for use by hardware for IMPLEMENTATION DEFINED usage. The value is:

`0x2` Hierarchical Permission Disables and Hardware allocation of bits[62:59] supported.

CNP, [15:12]

Common Not Private. Indicates support for selective sharing of TLB entries across multiple cores. The value is:

`0x1` CnP bit supported.

XNX, [11:8]

Execute Never. Indicates whether the stage 2 translation tables allows the stage 2 control of whether memory is executable at EL1 independent of whether memory is executable at EL0. The value is:

`0x1` EL0/EL1 execute control distinction at stage2 bit supported.

AC2, [7:4]

Indicates the extension of the ACTLR and HACTLR registers using ACTLR2 and HACTLR2. The value is:

`0x1` ACTLR2 and HACTLR2 are implemented.

SpecSEI, [3:0]

Describes whether the core can generate SError interrupt exceptions from speculative reads of memory, including speculative instruction fetches. The value is:

`0x0` The core never generates an SError interrupt due to an external abort on a speculative read.

Configurations

ID_MMFR4 is architecturally mapped to AArch64 register ID_MMFR4_EL1. See *B2.75 ID_MMFR4_EL1, AArch32 Memory Model Feature Register 4, EL1* on page B2-397.

There is one copy of this register that is used in both Secure and Non-secure states.

Must be interpreted with ID_MMFR0, ID_MMFR1, ID_MMFR2, and ID_MMFR3. See:

- *B1.66 ID_MMFR0, Memory Model Feature Register 0* on page B1-227.
- *B1.67 ID_MMFR1, Memory Model Feature Register 1* on page B1-229.
- *B1.68 ID_MMFR2, Memory Model Feature Register 2* on page B1-231.
- *B1.69 ID_MMFR3, Memory Model Feature Register 3* on page B1-233.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

B1.71 ID_PFR0, Processor Feature Register 0

The ID_PFR0 provides top-level information about the instruction sets supported by the core in AArch32.

Bit field descriptions

ID_PFR0 is a 32-bit register, and must be interpreted with ID_PFR1. It is part of the Identification registers functional group.

This register is Read Only.

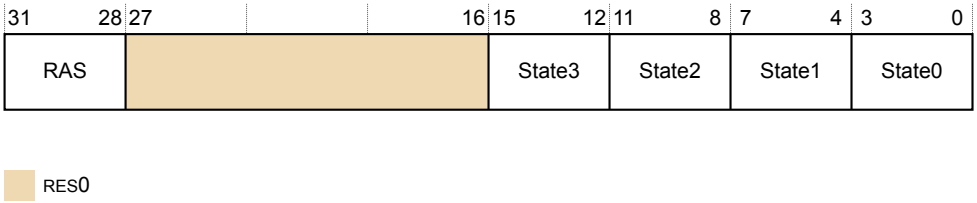

Figure B1-55 ID_PFR0 bit assignments

RAS, [31:28]
RAS extension version. The value is:

`0x1` Version 1 of the RAS extension is present.

RES0, [27:16]

RES0 Reserved.

State3, [15:12]
Indicates support for *Thumb Execution Environment* (T32EE) instruction set. This value is:

`0x0` Core does not support the T32EE instruction set.

State2, [11:8]
Indicates support for Jazelle. This value is:

`0x1` Core supports trivial implementation of Jazelle.

State1, [7:4]
Indicates support for T32 instruction set. This value is:

`0x3` Core supports T32 encoding after the introduction of Thumb-2 technology, and for all 16-bit and 32-bit T32 basic instructions.

State0, [3:0]
Indicates support for A32 instruction set. This value is:

`0x1` A32 instruction set implemented.

Configurations
ID_PFR0 is architecturally mapped to AArch64 register ID_PFR0_EL1. See *B2.76 ID_PFR0_EL1, AArch32 Processor Feature Register 0, EL1* on page B2-399.

There is one copy of this register that is used in both Secure and Non-secure states.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

B1.72 ID_PFR1, Processor Feature Register 1

The ID_PFR1 provides information about the programmers model and architecture extensions supported by the core.

Bit field descriptions

ID_PFR1 is a 32-bit register, and must be interpreted with ID_PFR0. It is part of the Identification registers functional group.

This register is Read Only.

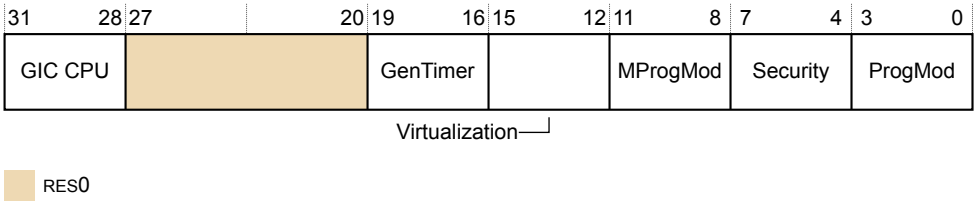

Figure B1-56 ID_PFR1 bit assignments

GIC CPU, [31:28]

GIC CPU support:

`0x0` GIC CPU interface is disabled, **GICCDISABLE** is HIGH.

`0x1` GIC CPU interface is enabled, **GICCDISABLE** is LOW.

RES0, [27:20]

RES0 Reserved.

GenTimer, [19:16]

Generic Timer support:

`0x1` Generic Timer is implemented.

Virtualization, [15:12]

Indicates support for Virtualization:

`0x1` The following Virtualization is implemented:

- The SCR.SIF bit.
- The modifications to the SCR.AW and SCR.FW bits described in the Virtualization Extensions.
- The MSR (Banked register) and MRS (Banked register) instructions.
- The ERET instruction.
- EL2, Hyp mode, the HVC instruction implemented.

MProgMod, [11:8]

M profile programmers model support:

`0x0` Not supported.

Security, [7:4]

Security support:

`0x1` The following Security items are implemented:

- The VBAR register.
- The TTBCR.PD0 and TTBCR.PD1 bits.
- The ability to access Secure or Non-secure physical memory is supported.
- EL3, Monitor mode, the SMC instruction implemented.

ProgMod, [3:0]

Indicates support for the standard programmers model for Armv4 and later.

Model must support User, FIQ, IRQ, Supervisor, Abort, Undefined and System modes:

`0x1` Supported.

Configurations

ID_PFR1 is architecturally mapped to AArch64 register ID_PFR1_EL1. See *B2.77 ID_PFR1_EL1, AArch32 Core Feature Register 1, EL1* on page B2-400.

There is one copy of this register that is used in both Secure and Non-secure states.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

B1.73 IFSR, Instruction Fault Status Register

The IFSR holds status information about the last instruction fault.

Bit field descriptions

IFSR is a 32-bit register, and is part of the Exception and fault handling registers functional group.

There are two formats for this register. The current translation table format determines which format of the register is used.

- *B1.73.1 IFSR with Short-descriptor translation table format* on page B1-240.
- *B1.73.2 IFSR with Long-descriptor translation table format* on page B1-240.

Configurations

IFSR (NS) is architecturally mapped to AArch64 register IFSR32_EL2. See *B2.78 IFSR32_EL2, Instruction Fault Status Register, EL2* on page B2-401.

If EL3 is using AArch32, there are separate Secure and Non-secure instances of this register.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

This section contains the following subsections:

- *B1.73.1 IFSR with Short-descriptor translation table format* on page B1-240.
- *B1.73.2 IFSR with Long-descriptor translation table format* on page B1-240.

B1.73.1 IFSR with Short-descriptor translation table format

IFSR has a specific format when using the Short-descriptor translation table format.

The following figure shows the IFSR bit assignments when using the Short-descriptor translation table format.

When TTBCR.EAE==0:

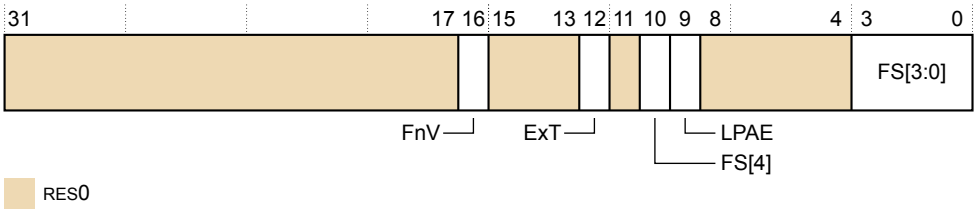

Figure B1-57 IFSR bit assignments for Short-descriptor translation table format

ExT, [12]

External abort type.

Read as zero.

For aborts other than external aborts, this bit always returns 0.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

B1.73.2 IFSR with Long-descriptor translation table format

IFSR has a specific format when using the Long-descriptor translation table format.

The following figure shows the IFSR bit assignments when using the Long-descriptor translation table format.

When TTBCR.EAE==1:

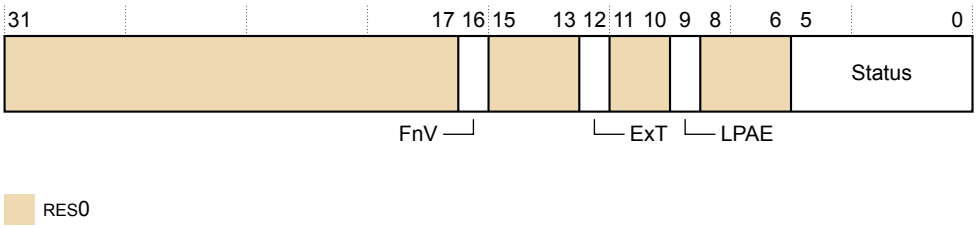

Figure B1-58 IFSR bit assignments for Long-descriptor translation table format

ExT, [12]

External abort type.

Read as zero.

For aborts other than external aborts, this bit always returns 0.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

B1.74 MIDR, Main ID Register

The MIDR provides identification information for the core, including an implementer code for the device and a device ID number.

Bit field descriptions

MIDR is a 32-bit register, and is part of the Identification registers functional group.

This register is Read Only.

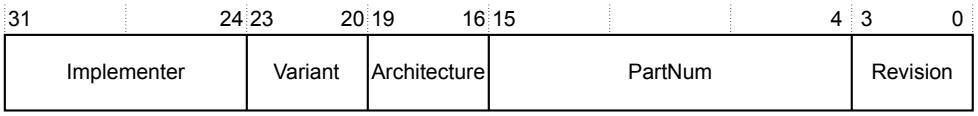

Figure B1-59 MIDR bit assignments

Implementer, [31:24]
Indicates the implementer code. This value is:

`0x41` ASCII character 'A' - implementer is Arm Limited.

Variant, [23:20]
Indicates the variant number of the core. This is the major revision number *n* in the *rn* part of the *rnpm* description of the product revision status. This value is:

`0x2` r2p1.

Architecture, [19:16]
Indicates the architecture code. This value is:

`0xF` Defined by ID registers.

PartNum, [15:4]
Indicates the primary part number. This value is:

`0xD0A` Cortex-A75 core.

Revision, [3:0]
Indicates the minor revision number of the core. This is the minor revision number *m* in the *pm* part of the *rnpm* description of the product revision status. This value is:

`0x1` r2p1.

Configurations
MIDR is:

- Architecturally mapped to the AArch64 MIDR_EL1 register. See *B2.83 MIDR_EL1, Main ID Register, EL1* on page B2-408.
- Architecturally mapped to external MIDR_EL1 register.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

B1.75 MPIDR, Multiprocessor Affinity Register

The MPIDR provides an additional core identification mechanism for scheduling purposes in a cluster. EDDEVAFF0 is a read-only copy of MPIDR accessible from the external debug interface.

Bit field descriptions

MPIDR is a 32-bit register, and is part of the Identification registers functional group.

This register is Read Only.

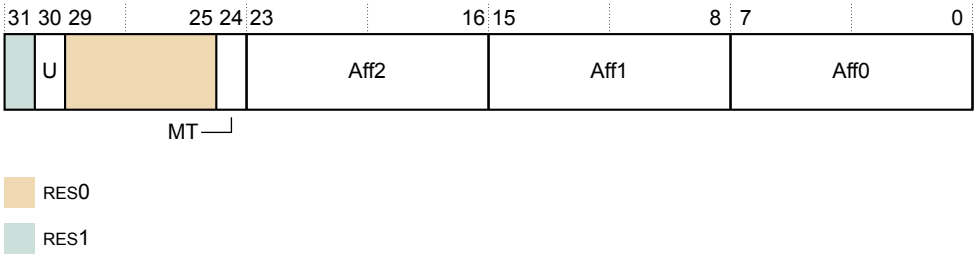

Figure B1-60 MPIDR bit assignments

RES1, [31]

RES1.

U, [30]

Indicates a uniprocessor system, as distinct from core 0 in a multiprocessor system. This value is:

`0b0` Core is part of a multiprocessor system.

RES0, [29:25]

RES0 Reserved.

MT, [24]

Indicates whether the lowest level of affinity consists of logical cores that are implemented using a multi-threading type approach. This value is:

`0b1` Affinity 0 represents threads. However, Cortex-A75 is not multithreaded and so affinity 0 will always be zero. This allows consistency when in a system with other cores that are multithreaded.

Aff2, [23:16]

Affinity level 2. This level of affinity identifies different clusters within the system. The value in this field is equal to the value present on the **CLUSTERIDFAFF2** configuration signal.

Aff1, [15:8]

Affinity level 1. This level of affinity identifies individual cores within the local DynamIQ cluster. The value can range from `0x00` for core 0, to `0x07` for core 7.

Aff0, [7:0]

Affinity level 0. The level identifies individual threads within a multi-threaded core. The Cortex-A75 core is single-threaded, so this field has the value `0x00`.

Configurations

The MPIDR is:

- Architecturally mapped to the AArch64 MPIDR_EL1[31:0] register. See *B2.84 MPIDR_EL1, Multiprocessor Affinity Register, EL1* on page B2-409.
- Mapped to external EDDEVAFF0 register.

There is one copy of this register that is used in both Secure and Non-secure states.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

B1.76 MVBAR, Monitor Vector Base Address Register

The MVBAR holds the vector base address for any exception that is taken to Monitor mode when EL3 is implemented and can use AArch32.

Bit field descriptions

MVBAR is a 32-bit register.

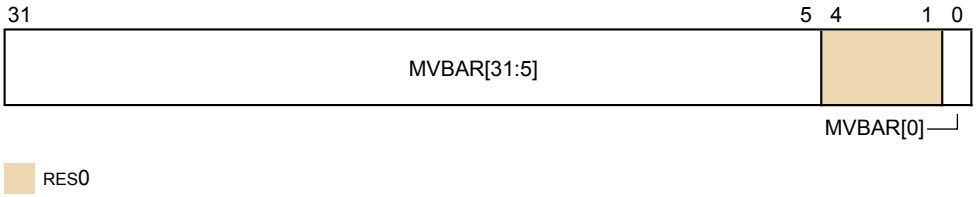

Figure B1-61 MVBAR bit assignments

MVBAR[31:5]

If the input signal **vinithi_i** is at 0, this field resets to `0x00000000`.

If the input signal **vinithi_i** is at 1, this field resets to `0xFFFF0000`.

MVBAR[4:1]

Reserved, RES0.

MVBAR[0]

This bit resets to 0.

Configurations

This register is only accessible in Secure state.

Write access to MVBAR is disabled when the CP15SDISABLE signal is asserted HIGH.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

B1.77 NSACR, Non-Secure Access Control Register

The NSACR defines the Non-secure access permissions to Trace, Advanced SIMD and floating-point functionality when EL3 is using AArch32.

Bit field descriptions

NSACR is a 32-bit register, and is part of the Security registers functional group.

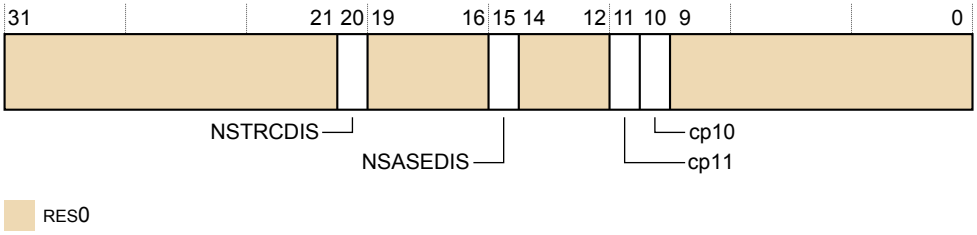

Figure B1-62 NSACR bit assignments

NSTRCDIS, [20]
Disable Non-secure System register accesses to all implemented trace registers:

RAZ/WI.

RES0, [19:16]

RES0 Reserved.

NSASEDIS, [15]
Disable Non-secure Advanced SIMD functionality:

`0` This bit has no effect on the ability to write CPACR.ASEDIS. This is the reset value.

`1` When executing in Non-secure state, the CPACR.ASEDIS bit has a fixed value of 1 and writes to it are ignored.

If Advanced SIMD and floating-point are not implemented, this bit is RES0.

RES0, [14:12]

RES0 Reserved.

cp11, [11]
This bit resets to 0.

cp10, [10]
This bit resets to 0.

RES0, [9:0]

RES0 Reserved.

Configurations
There is one copy of this register that is used in both Secure and Non-secure states.

If EL3 is using AArch64, then any reads of the NSACR from Non-secure EL2 or Non-secure EL1 using AArch32 return a fixed value of `0x00000C00`.

In AArch64, the NSACR functionality is replaced by the behavior in CPTR_EL3.

Any read or write to NSACR in Secure EL1 state in AArch32 is trapped as an exception to EL3.

B1.78 PAR, Physical Address Register

The PAR returns the *output address* (OA) from an address translation instruction that executed successfully, or fault information if the instruction did not execute successfully.

Configuration Details

PAR is a 64-bit register that can also be accessed as a 32-bit value. If it is accessed as a 32-bit register, accesses read and write bits[31:0] and do not modify bits[63:32].

PAR is part of the Address translation instructions functional group.

Configurations

AArch32 System register PAR is architecturally mapped to AArch64 System register PAR_EL1. See *B2.85 PAR_EL1, Physical Address Register, EL1* on page B2-411.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

This section contains the following subsections:

- *B1.78.1 PAR with Short-descriptor translation table format* on page B1-247.
- *B1.78.2 PAR with Long-descriptor translation table format* on page B1-248.

B1.78.1 PAR with Short-descriptor translation table format

PAR details when the core is using the Short-descriptor translation table format.

F, [0]

Indicates whether the instruction performed a successful address translation.

`0` Address translation completed successfully.

`1` Address translation aborted.

Bit field descriptions, PAR.F is 0

The following figure shows the PAR bit assignments when PAR.F is 0.

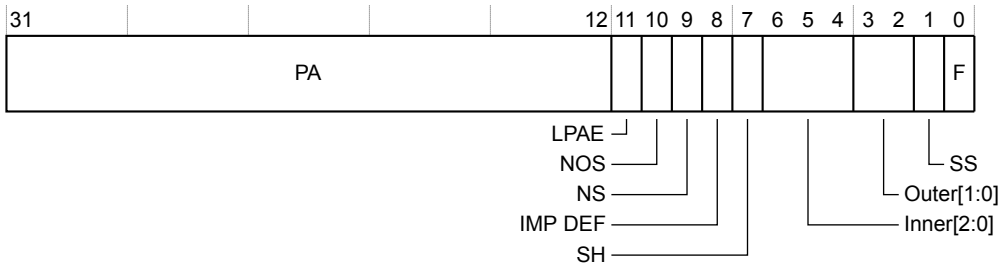

Figure B1-63 PAR bit assignments, PAR.F is 0

NOS, [10]

Not Outer Shareable. When the returned value of PAR.SH is 1, indicates the Shareability attribute for the physical memory region:

`0`

Memory region is Outer Shareable.

IMP DEF, [8]

IMPLEMENTATION DEFINED. Bit[8] is RES0.

Bit field descriptions, PAR.F is 1

When PAR.F==1, IMPLEMENTATION DEFINED bits[31:16] are RES0.

For other details, see the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

B1.78.2 PAR with Long-descriptor translation table format

PAR details when the PE is using the Long-descriptor translation table format.

F, [0]

Indicates whether the instruction performed a successful address translation.

`0` Address translation completed successfully.

`1` Address translation aborted.

Bit field descriptions, PAR.F is 0

The following figure shows the PAR bit assignments when PAR.F is 0.

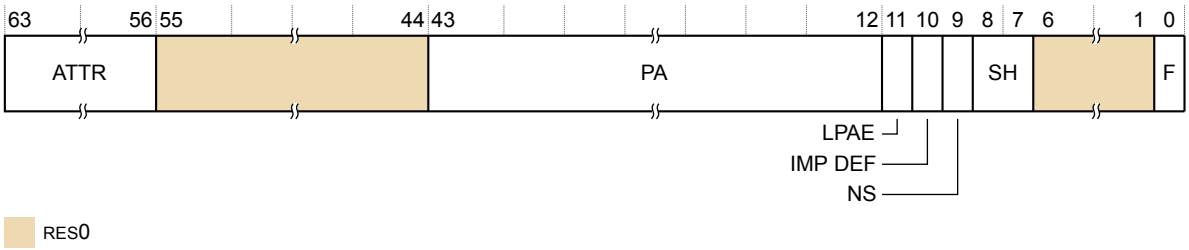

Figure B1-64 PAR bit assignments, PAR_EL1.F is 0

IMP DEF, [10]

IMPLEMENTATION DEFINED. Bit[10] is RES0.

Bit field descriptions, PAR.F is 1

When PAR.F==1, IMPLEMENTATION DEFINED bits[63:48] are RES0.

For other details, see the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

B1.79 REVIDR, Revision ID Register

The REVIDR provides revision information, additional to that in the MIDR, which identifies minor fixes (errata) which may be present in a specific implementation of the Cortex-A75 core.

Bit field descriptions

REVIDR is a 32-bit register, and is part of the Identification registers functional group.

This register resets to value `0x00000000`.

This register is Read Only.

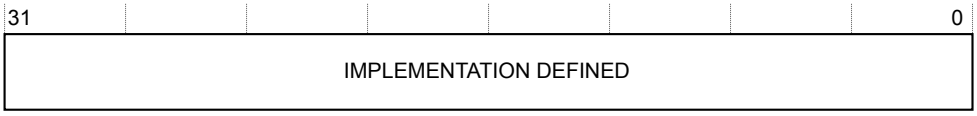

Figure B1-65 REVIDR bit assignments

IMPLEMENTATION DEFINED, [31:0]

Implementation Defined.

Configurations

AArch32 System register REVIDR is architecturally mapped to AArch64 System register REVIDR_EL1. See *B2.86 REVIDR_EL1, Revision ID Register, EL1* on page B2-412.

There is one instance of this register that is used in both Secure and Non-secure states.

B1.80 RVBAR, Reset Vector Base Address Register

The RVBAR contains the IMPLEMENTATION DEFINED address that execution starts from after reset when executing in AArch32 state if EL3 is not implemented.

Bit field descriptions

RVBAR is a 32-bit register.

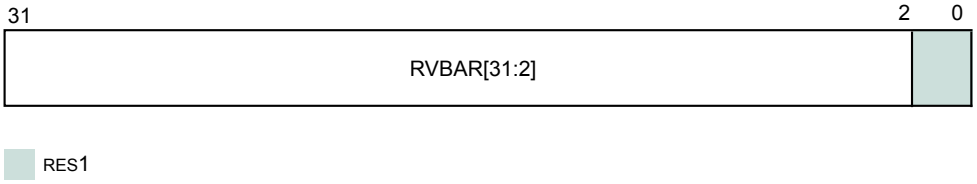

Figure B1-66 RVBAR bit assignments

RVBAR[31:2]
Bits[31:2] reset to the input signal **rvbaraddr_i** .

Configurations
There is one instance of this register that is used in both Secure and Non-secure states.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

B1.81 SCR, Secure Configuration Register

The SCR defines the configuration of the current Security state when EL3 is implemented and can use AArch32.

It specifies:

- The Security state, either Secure or Non-secure.
- What mode the core branches to if an IRQ, FIQ, or External Abort occurs.
- Whether the CPSR.F or CPSR.A bits can be modified when SCR.NS==1.

Bit field descriptions

SCR is a 32-bit register.

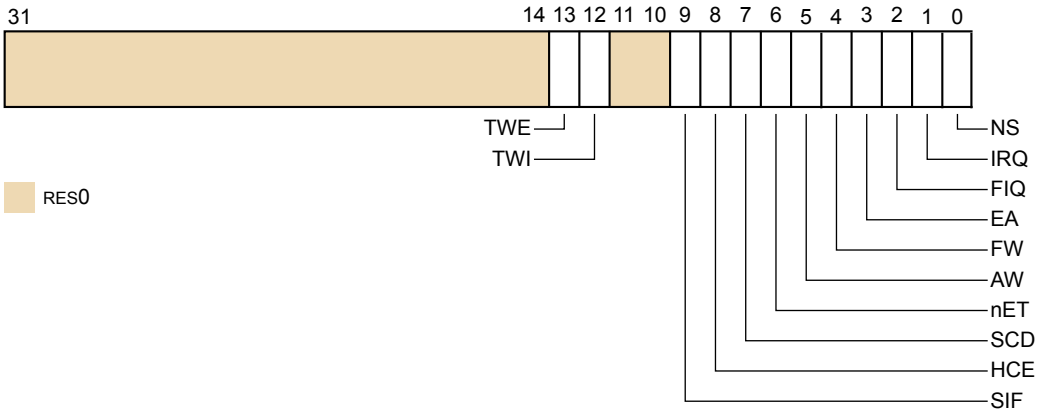

Figure B1-67 SCR bit assignments

This register resets to value `0x00000000`.

Configurations

This register is only accessible in Secure state.

AArch32 System register SCR can be mapped to AArch64 System register SCR_EL3, but this is not architecturally mandated.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

B1.82 SCTLR, System Control Register

The SCTLR provides the top-level control of the system, including its memory system.

Bit field descriptions

SCTLR is a 32-bit register, and is part of the Other system control registers functional group.

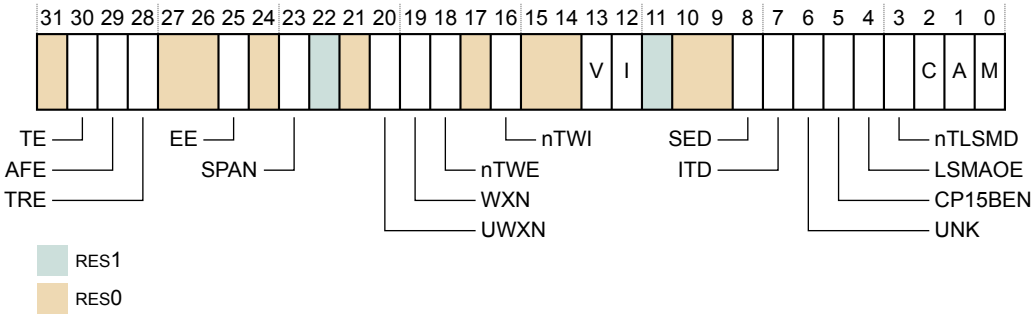

Figure B1-68 SCTLR bit assignments

TE, [30]
T32 Exception enable.

This field resets to a value determined by the input configuration signal **cfgte_i**.

AFE, [29]
Access Flag Enable.

This field resets to 0.

TRE, [28]
TEX remap enable.

This field resets to 0.

EE, [25]
Exception Endianness bit.

This field resets to a value determined by the input configuration signal **cfgend_i**.

SPAN, [23]

This field resets to 1.

UWXN, [20]
Unprivileged write permission implies PL1 XN (Execute-never).

This field resets to 0.

WXN, [19]
Write permission implies XN (Execute-never).

This field resets to 0.

nTWE, [18]
Traps PL0 execution of WFE instructions to Undefined mode.

This field resets to 1.

nTWI, [16]

Traps PL0 execution of WFI instructions to Undefined mode.

This field resets to 1.

V, [13]

Vectors bit.

This field resets to a value determined by the input configuration signal **vinithi_i**.

I, [12]

Instruction access Cacheability control, for accesses at EL1 and EL0.

This field resets to 0.

SED, [8]

SETEND instruction disable. Disables SETEND instructions at PL0 and PL1:

`0` SETEND instruction execution is enabled at PL0 and PL1.

`1` SETEND instructions are UNDEFINED at PL0 and PL1.

This field resets to 0.

ITD, [7]

RES0 All IT instruction functionality is always implemented in PL0, PL1 and enabled at PL2.

UNK, [6]

Writes to this bit are IGNORED. Reads of this bit return an UNKNOWN value.

CP15BEN, [5]

System instruction memory barrier enable. Enables accesses to the DMB, DSB, and ISB System instructions in the (coproc==1111) encoding space from PL1 and PL0.

`0` PL0 and PL1 execution of the CP15DMB, CP15DSB, and CP15ISB instructions is UNDEFINED.

`1` PL0 and PL1 execution of the CP15DMB, CP15DSB, and CP15ISB instructions is enabled.

This field resets to 1.

LSMAOE, [4]

Load/Store Multiple Atomicity and Ordering Enable.

RES1 This bit is not controllable. The ordering and interrupt behavior of Load/Store Multiple is as defined for Armv8.

nTLSMD, [3]

no Trap Load/Store Multiple to Device-nGRE/Device-nGnRE/Device-nGnRnE memory.

RES1 This bit is not controllable. Load/Store Multiple to memory marked at stage1 as Device-nGRE/Device-nGnRE/Device-nGnRnE memory does not generate a stage 1 alignment fault as a result of this mechanism.

C, [2]

Cacheability control, for data accesses at EL1 and EL0.

This field resets to 0.

A, [1]

Alignment check enable.

This field resets to 0.

M, [0]

MMU enable for EL1 and EL0 stage 1 address translation.

This field resets to 0.

Configurations

SCTLR (NS) is architecturally mapped to AArch64 register SCTLR_EL1. See *B2.89 SCTLR_EL1, System Control Register, EL1* on page B2-416.

If EL3 is using AArch32, there are separate Secure and Non-secure instances of this register.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

B1.83 SDCR, Secure Debug Control Register

The SDCR Controls debug and performance monitors functionality in Secure state.

Bit field descriptions

SDCR is a 32-bit register, and is part of:

- The Debug registers functional group.
- The Security registers functional group.

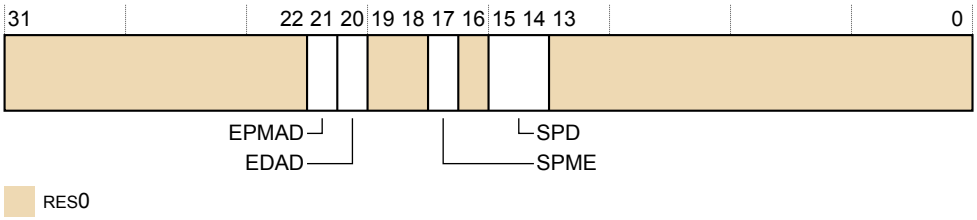

Figure B1-69 SDCR bit assignments

EPMAD, [21]

External debugger access to Performance Monitors registers disabled. This disables access to these registers by an external debugger:

`0b0` Access to Performance Monitors registers from external debugger is permitted. This is the reset value.

`0b1` Access to Performance Monitors registers from external debugger is disabled, unless overridden by authentication interface.

EDAD, [20]

External debugger access to breakpoint and watchpoint registers disabled. This disables access to these registers by an external debugger:

`0b0` Access to breakpoint and watchpoint registers from external debugger is permitted. This is the reset value.

`0b1` Access to breakpoint and watchpoint registers from external debugger is disabled, unless overridden by authentication interface.

SPME, [17]

Secure performance monitors enable. This allows event counting in Secure state:

`0b0` Event counting prohibited in Secure state. This is the reset value.

`0b1` Event counting allowed in Secure state.

SPD, [15:14]

AArch32 Secure privileged debug. Enables or disables debug exceptions from Secure state, other than Breakpoint Instruction exceptions. Valid values for this field are:

`0b00` Legacy mode. Debug exceptions from Secure EL1 are enabled by the authentication interface. This is the reset value.

`0b10` Secure privileged debug disabled. Debug exceptions from Secure EL1 are disabled.

`0b11` Secure privileged debug enabled. Debug exceptions from Secure EL1 are enabled.

Configurations

SDCR is mapped to AArch64 register MDCR_EL3. See *B2.82 MDCR_EL3, Monitor Debug Configuration Register, EL3* on page B2-406.

The SDCR is only accessible in Secure state.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

B1.84 TTBCR, Translation Table Base Control Register

The TTBCR determines which of the Translation Table Base Registers defines the base address for a translation table walk required for the stage 1 translation of a memory access from any mode other than Hyp mode

It also controls the translation table format and, when using the Long-descriptor translation table format, holds cacheability and shareability information.

Bit field descriptions

TTBCR is a 32-bit register.

TTBCR can be used with the Long-descriptor translation table format or with the Short-descriptor translation table format. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile* for details.

If EL3 is implemented and is using AArch32, there are separate Secure and Non-secure instances of this register.

Field EAE, bit[31], resets to 0 for both Secure and Non-secure instanced.

The entire Secure instance of this register resets to `0x00000000`.

Configurations

AArch32 System register TTBCR is architecturally mapped to AArch64 System register TCR_EL1[31:0].

The current translation table format determines which format of the register is used.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

B1.85 TTBR0, Translation Table Base Register 0

The TTBR0 holds the base address of translation table 0, and information about the memory it occupies. This is one of the translation tables for the stage 1 translation of memory accesses from modes other than Hyp mode.

Usage constraints

TTBR0 is part of the Virtual memory control registers functional group.

There are two formats for this register. TTBCR.EAE determines which format of the register is used.

- *B1.85.1 TTBR0 with Short-descriptor translation table format* on page B1-258.
- *B1.85.2 TTBR0 with Long-descriptor translation table format* on page B1-259.

Configurations

TTBR0 (NS) is architecturally mapped to AArch64 register TTBR0_EL1. See *B2.95 TTBR0_EL1, Translation Table Base Register 0, EL1* on page B2-424.

TTBR0 (S) is mapped to AArch64 register TTBR0_EL3. See *B2.97 TTBR0_EL3, Translation Table Base Register 0, EL3* on page B2-426.

If EL3 is using AArch32, there are separate Secure and Non-secure instances of this register.

Attributes

TTBR0 is a 64-bit register that can also be accessed as a 32-bit value. If it is accessed as a 32-bit register, accesses read and write bits [31:0] and do not modify bits [63:32].
TTBCR.EAE determines which TTBR0 format is used:

- EAE==0:32-bit format is used. TTBR0[63:32] are ignored.
- EAE==1:64-bit format is used.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

This section contains the following subsections:

- *B1.85.1 TTBR0 with Short-descriptor translation table format* on page B1-258.
- *B1.85.2 TTBR0 with Long-descriptor translation table format* on page B1-259.

B1.85.1 TTBR0 with Short-descriptor translation table format

TTBR0 has a specific format when using the Short-descriptor translation table format. TTBCR.EAE determines which format of the register is in use.

Bit field descriptions

The following figure shows the TTBR0 bit assignments when TTBCR.EAE is 0.

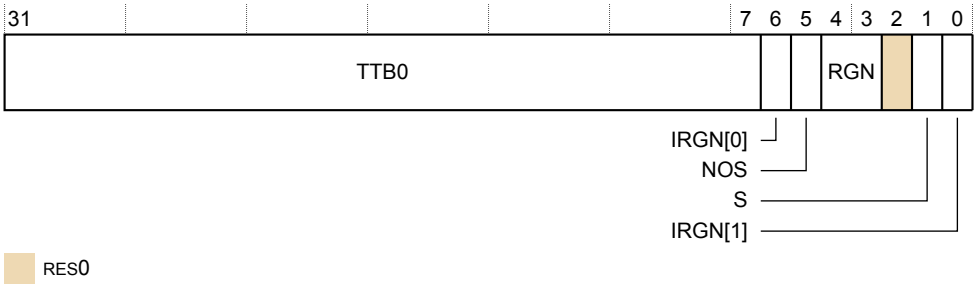

Figure B1-70 TTBR0 bit assignments, TTBCR.EAE is 0

IMP, [2]

RES0 Reserved.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

B1.85.2 TTBR0 with Long-descriptor translation table format

TTBR0 has a specific format when using the Long-descriptor translation table format. TTBCR.EAE determines which format of the register is in use.

The Long-descriptor translation table format for TTBR0 is architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile* for a full description.

Bit field descriptions

The following bit is specific to the implementation:

CnP, [0]

Common not private. The possible values are:

0 CnP is not supported.

1 CnP is supported.

B1.86 TTBR1, Translation Table Base Register 1

The TTBR1 holds the base address of translation table 1, and information about the memory it occupies. This is one of the translation tables for the stage 1 translation of memory accesses from modes other than Hyp mode.

Usage constraints

TTBR1 is part of the Virtual memory control registers functional group.

There are two formats for this register. TTBCR.EAE determines which format of the register is used.

- *B1.86.1 TTBR1 with Short-descriptor translation table format* on page B1-260.
- *B1.86.2 TTBR1 with Long-descriptor translation table format* on page B1-261.

Configurations

TTBR1 (NS) is architecturally mapped to AArch64 register TTBR1_EL1. See *B2.98 TTBR1_EL1, Translation Table Base Register 1, EL1* on page B2-427.

If EL3 is using AArch64, there is a single instance of this register.

Attributes

TTBR1 is a 64-bit register that can also be accessed as a 32-bit value. If it is accessed as a 32-bit register, accesses read and write bits [31:0] and do not modify bits [63:32].
TTBCR.EAE determines which TTBR1 format is used:

- EAE==0:32-bit format is used. TTBR1[63:32] are ignored.
- EAE==1:64-bit format is used.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

This section contains the following subsections:

- *B1.86.1 TTBR1 with Short-descriptor translation table format* on page B1-260.
- *B1.86.2 TTBR1 with Long-descriptor translation table format* on page B1-261.

B1.86.1 TTBR1 with Short-descriptor translation table format

TTBR1 has a specific format when using the Short-descriptor translation table format. TTBCR.EAE determines which format of the register is in use.

Bit field descriptions

The following figure shows the TTBR1 bit assignments when TTBCR.EAE is 0.

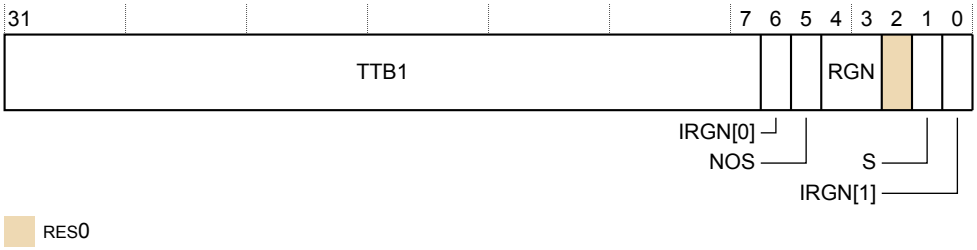

Figure B1-71 TTBR1 bit assignments, TTBCR.EAE is 0

IMP, [2]

RES0 Reserved.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

B1.86.2 TTBR1 with Long-descriptor translation table format

TTBR1 has a specific format when using the Long-descriptor translation table format. TTBCR.EAE determines which format of the register is in use.

The Long-descriptor translation table format for TTBR1 is architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile* for a full description.

Bit field descriptions

The following bit is specific to the implementation:

CnP, [0]

Common not private. The possible values are:

0 CnP is not supported.

1 CnP is supported.

B1.87 VBAR, Vector Base Address Register

The VBAR holds the exception base address for exceptions that are not taken to Monitor mode or to Hyp mode when high exception vectors are not selected.

Bit field descriptions

VBAR is a 32-bit register, and is part of the Exception and fault handling registers functional group.

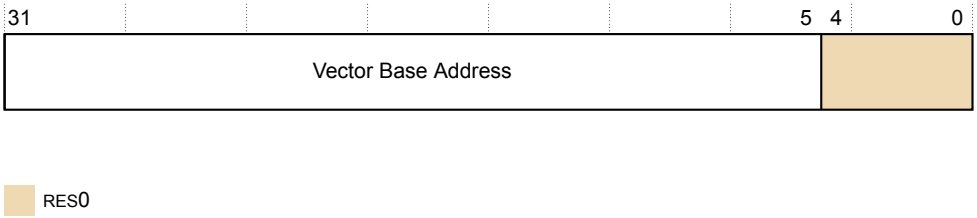

Figure B1-72 VBAR bit assignments

This register resets to an UNKNOWN value.

Configurations

If EL3 is using AArch32, there are separate Secure and Non-secure instances of this register.

The Non-secure VBAR is architecturally mapped to the AArch64 VBAR_EL1 register.

The Secure VBAR is mapped to AArch64 register VBAR_EL3[31:0].

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

B1.88 VDFSR, Virtual SError Exception Syndrome Register

The VDFSR provides the syndrome value reported to software on taking a virtual SError interrupt exception.

Bit field descriptions

VDFSR is a 32-bit register, and is part of :

- The Exception and fault handling registers functional group.
- The Virtualization registers functional group.

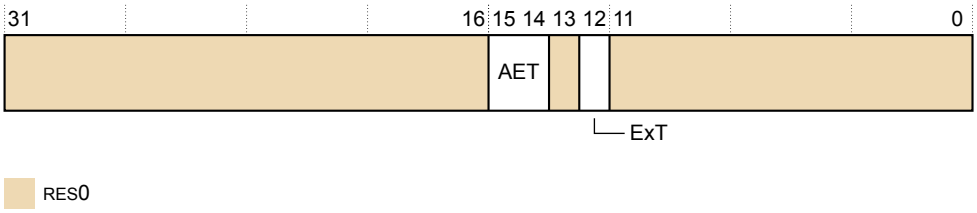

Figure B1-73 VDFSR bit assignments

RES0, [31:16]

RES0 Reserved.

AET, [15:14]

Asynchronous Error Type. Describes the state of the core after taking a virtual SError interrupt exception. Software might use the information in the syndrome registers to determine what recovery might be possible. The value is:

RES0 The core is always in *Uncontainable* (UC) state when an SEI is signaled.

RES0, [13]

RES0 Reserved.

ExT, [12]

External abort type.

RES0.

This register is only used for external aborts.

RES0, [11:0]

RES0 Reserved.

Configurations

AArch32 System register VDFSR is architecturally mapped to AArch64 System register VSESR_EL2 [31:0]. See *B2.101 VSESR_EL2, Virtual SError Exception Syndrome Register* on page B2-432.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

B1.89 VDISR, Virtual Deferred Interrupt Status Register

The VDISR records that a virtual SError interrupt has been consumed by an ESB instruction executed at Non-secure EL1.

Bit field descriptions

VDISR is a 32-bit register, and is part of *Reliability, Availability, Serviceability* (RAS) registers functional group.

There are two formats for this register. The current translation table format determines which format of the register is used:

- When written at EL1 using short-descriptor format. See *B1.89.1 VDISR with Short-descriptor translation table format* on page B1-264
- When written at EL1 using long-descriptor format. See *B1.89.2 VDISR with Long-descriptor translation table format* on page B1-265

Configurations

There is one instance of VDISR that is used in both Secure and Non-secure states.
Present only if all of the following are present and is UNDEFINED otherwise:

- EL2 is implemented and using AArch32.
- The RAS extension is implemented.

If the highest implemented Exception level is using AArch64, AArch32 System register VDISR is architecturally mapped to AArch64 System register VDISR_EL2. See *B2.100 VDISR_EL2, Virtual Deferred Interrupt Status Register, EL2* on page B2-429.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

This section contains the following subsections:

- *B1.89.1 VDISR with Short-descriptor translation table format* on page B1-264.
- *B1.89.2 VDISR with Long-descriptor translation table format* on page B1-265.

B1.89.1 VDISR with Short-descriptor translation table format

VDISR has a specific format when written at EL1 using the Short-descriptor translation table format.

Bit field descriptions

The following figure shows the VDISR bit assignments when using the Short-descriptor translation table format.

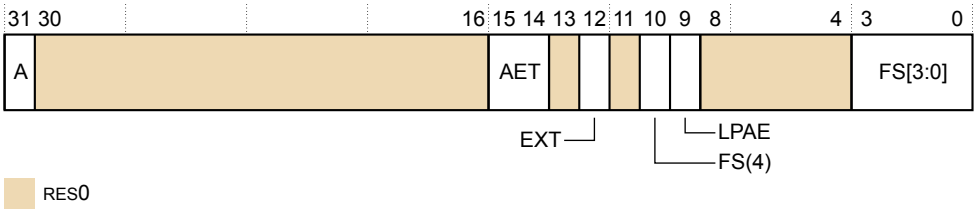

Figure B1-74 VDISR bit assignments for Short-descriptor translation table format

A, [31]

Set to 1 when ESB defers a virtual SError interrupt.

RES0, [30:16]

RES0 Reserved.

AET, [15:14]

Asynchronous Error Type. Describes the state of the PE after taking an asynchronous Data Abort exception. The value is:

`0b00` *Uncorrected error, Uncontainable* (UC).

RES0, [13]

RES0 Reserved.

EXT, [12]

External Abort Type. This bit is defined as RES0.

RES0, [11]

RES0 Reserved.

FS(4), [10,3:0]

Fault status code. Set to `0b10110` when ESB defers a virtual SError interrupt. The value of this field is:

`0b10110`
Asynchronous SError interrupt.

LPAE, [9]

Format. The value is:

`0b0`
Using the Short-descriptor translation table format.

RES0, [8:4]

RES0 Reserved.

B1.89.2 VDISR with Long-descriptor translation table format

VDISR has a specific format when written at EL1 using the Long-descriptor translation table format.

Bit field descriptions

The following figure shows the VDISR bit assignments when using the Long-descriptor translation table format.

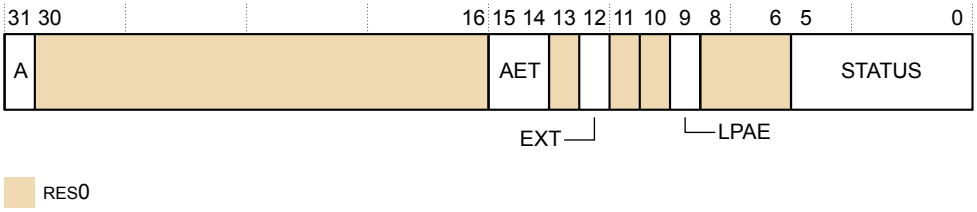

Figure B1-75 VDISR bit assignments for Long-descriptor translation table format

A, [31]

Set to 1 when ESB defers a virtual SError interrupt.

RES0, [30:16]

RES0 Reserved.

AET, [15:14]

Asynchronous Error Type. Describes the state of the PE after taking an asynchronous Data Abort exception. The value is:

`0b00` *Uncorrected error, Uncontainable* (UC).

RES0, [13]

RES0 Reserved.

EXT, [12]

External Abort Type. This bit is defined as RES0.

RES0, [11]

RES0 Reserved.

RES0, [10]

RES0 Reserved.

LPAE, [9]

Format. The value is:

`0b1`

Using the Long-descriptor translation table format.

RES0, [8:6]

RES0 Reserved.

STATUS, [5:0]

Fault status code. Set to `0b010001` when ESB defers a virtual SError interrupt. The value of this field is:

`0b010001`

Asynchronous SError interrupt.

B1.90 VMPIDR, Virtualization Multiprocessor ID Register

The VMPIDR provides the value of the Virtualization Multiprocessor ID. This is the value returned by Non-secure EL1 reads of MPIDR.

Configurations

VMPIDR is architecturally mapped to AArch64 register VMPIDR_EL2[31:0].

This register is accessible only at EL2 or EL3.

This register resets to MPIDR value.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

B1.91 VPIDR, Virtualization Processor ID Register

The VPIDR holds the value of the Virtualization Processor ID. This is the value returned by Non-secure EL1 reads of MIDR.

Configurations

VPIDR is architecturally mapped to AArch64 register VPIDR_EL2.

This register resets to MIDR value.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

B1.92 VTCR, Virtualization Translation Control Register

The VTCR controls the translation table walks required for the stage 2 translation of memory accesses from Non-secure modes other than Hyp mode.

It also holds cacheability and shareability information for the accesses.

Bit field descriptions

VTCR is a 32-bit register, and is part of:

- The Virtualization registers functional group.
- The Virtual memory control registers functional group.

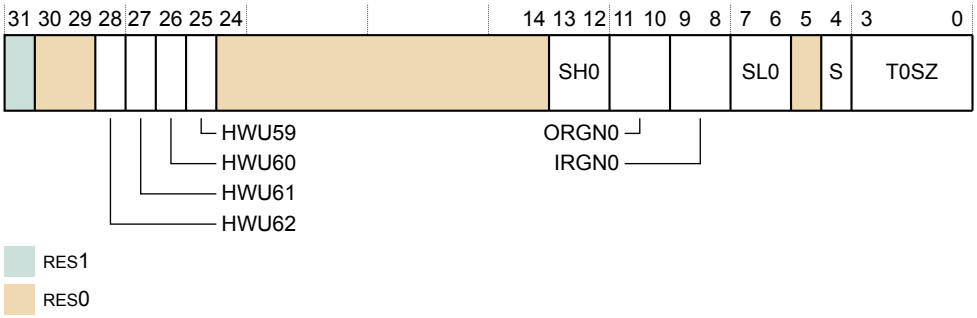

Figure B1-76 VTCR bit assignments

RES1, [31]

RES1 Reserved.

RES0, [30:29]

RES0 Reserved.

HWU62, [28]

Indicates implementation defined hardware use of bit[62] of the stage 2 translation table block or level 3 entry.

0b0 The associated stage 2 translation table block or level 3 entry bit cannot be interpreted by hardware for an implementation defined purpose.

0b1 The associated stage 2 translation table block or level 3 entry bit can be interpreted by hardware for an implementation defined purpose.

This bit is RAZ/WI.

HWU61, [27]

Indicates implementation defined hardware use of bit[61] of the stage 2 translation table block or level 3 entry.

0b0 The associated stage 2 translation table block or level 3 entry bit cannot be interpreted by hardware for an implementation defined purpose.

0b1 The associated stage 2 translation table block or level 3 entry bit can be interpreted by hardware for an implementation defined purpose.

This bit is RAZ/WI.

HWU60, [26]

Indicates implementation defined hardware use of bit[60] of the stage 2 translation table block or level 3 entry.

`0b0` The associated stage 2 translation table block or level 3 entry bit cannot be interpreted by hardware for an implementation defined purpose.

`0b1` The associated stage 2 translation table block or level 3 entry bit can be interpreted by hardware for an implementation defined purpose.

This bit is RAZ/WI.

HWU59, [25]

Indicates implementation defined hardware use of bit[59] of the stage 2 translation table block or level 3 entry.

`0b0` The associated stage 2 translation table block or level 3 entry bit cannot be interpreted by hardware for an implementation defined purpose.

`0b1` The associated stage 2 translation table block or level 3 entry bit can be interpreted by hardware for an implementation defined purpose.

This bit is RAZ/WI.

RES0, [24:14]

RES0 Reserved.

SH0, [13:12]

Shareability attribute for memory associated with translation table walks using TTBR0.

`0b00` Non-shareable.

`0b01` Reserved.

`0b10` Outer Shareable.

`0b11` Inner Shareable.

ORGN0, [11:10]

Outer cacheability attribute for memory associated with translation table walks using TTBR0.

`0b00` Normal memory, Outer Non-cacheable.

`0b01` Normal memory, Outer Write-Back Write-Allocate Cacheable.

`0b10` Normal memory, Outer Write-Through Cacheable.

`0b11` Normal memory, Outer Write-Back no Write-Allocate Cacheable.

IRGN0, [9:8]

Inner cacheability attribute for memory associated with translation table walks using TTBR0.

`0b00` Normal memory, Inner Non-cacheable.

`0b01` Normal memory, Inner Write-Back Write-Allocate Cacheable.

`0b10` Normal memory, Inner Write-Through Cacheable.

`0b11` Normal memory, Inner Write-Back no Write-Allocate Cacheable.

SL0, [7:6]

Starting level for translation table walks using VTTBR:

`0b00` Start at second level.

`0b01` Start at first level.

RES0, [5]

RES0 Reserved.

S, [4]

Sign extension bit. This bit must be programmed to the value of T0SZ[3]. If it is not, then the stage 2 T0SZ value is treated as an UNKNOWN value within the legal range that can be programmed.

T0SZ, [3:0]

The size offset of the memory region addressed by TTBR0. The region size is $2^{32-T0SZ}$ bytes.

Configurations

VTCR is architecturally mapped to AArch64 register VTCR_EL2. See *B2.102 VTCR_EL2, Virtualization Translation Control Register, EL2* on page B2-434.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

B1.93 VTTBR, Virtualization Translation Table Base Register

VTTBR holds the base address of the translation table for the stage 2 translation of memory accesses from Non-secure modes other than Hyp mode.

Bit field descriptions

VTTBR is a 64-bit register, and is part of:

- The Virtualization registers functional group.
- The Virtual memory control registers functional group.

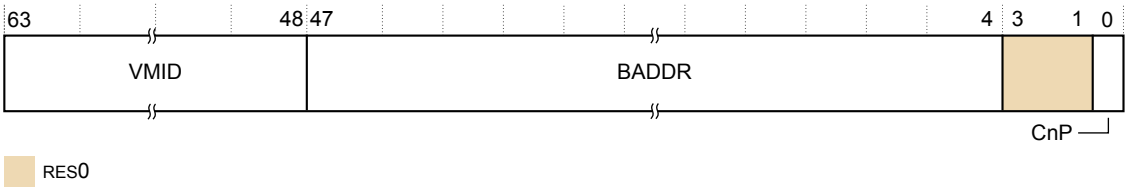

Figure B1-77 VTTBR bit assignments

VMID, [63:48]

This bit field resets to 0.

CnP, [0]

Common not Private. The value possible values are:

0 CnP is not supported.

1 CnP is supported.

Configurations

VTTBR is architecturally mapped to AArch64 register VTTBR_EL1. See *B2.103 VTTBR_EL2, Virtualization Translation Table Base Register, EL2* on page B2-435.

This register is used with the VTCR.

Some or all RW fields of this register have defined reset values. These apply only if the core resets into EL2 with EL2 using AArch32, or into EL3 with EL3 using AArch32. Otherwise, RW fields in this register reset to architecturally UNKNOWN values.

Bit fields and details that are not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

Chapter B2
AArch64 system registers

This chapter describes the system registers in the AArch64 state.

It contains the following sections:

B2.1 AArch64 registers

This chapter provides information about AArch64 system registers with implementation defined bit fields and implementation defined registers associated with the core.

The chapter provides implementation specific information, for a complete description of the registers, see the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The chapter is presented as follows:

AArch64 architectural system register summary
This section identifies the AArch64 architectural system registers implemented in the Cortex-A75 core that have implementation defined bit fields. The register descriptions for these registers only contain information about the implementation defined bits is described.

AArch64 implementation defined register summary
This section identifies the AArch64 architectural registers implemented in the Cortex-A75 core that are implementation defined.

AArch64 registers by functional group
This section groups the implementation defined registers and architectural system registers with implementation defined bit fields, as identified previously, by function. It also provides reset details for key register types.

Register descriptions
The remainder of the chapter provides register descriptions of the implementation defined registers and architectural system registers with implementation defined bit fields, as identified previously. These are listed in alphabetic order.

B2.2 AArch64 architectural system register summary

This section describes the AArch64 architectural system registers implemented in the Cortex-A75 core.

The section contains two tables:

Registers with implementation defined bit fields

This table identifies the architecturally defined registers in Cortex-A75 that have implementation defined bit fields. The register descriptions for these registers only contain information about the implementation defined bits.

See *Table B2-1 Registers with implementation defined bit fields* on page B2-277.

Other architecturally defined registers

This table identifies the other architecturally defined registers that are implemented in the Cortex-A75 core. These registers are described in the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

See *Table B2-2 Other architecturally defined registers* on page B2-281.

Table B2-1 Registers with implementation defined bit fields

Name	Op0	CRn	Op1	CRm	Op2	Width	Description
ACTLR_EL1	3	c1	0	c0	1	64	*B2.5 ACTLR_EL1, Auxiliary Control Register, EL1* on page B2-293
ACTLR_EL2	3	c1	4	c0	1	64	*B2.6 ACTLR_EL2, Auxiliary Control Register, EL2* on page B2-294
ACTLR_EL3	3	c1	6	c0	1	64	*B2.7 ACTLR_EL3, Auxiliary Control Register, EL3* on page B2-296
AIDR_EL1	3	c0	1	c0	7	32	*B2.14 AIDR_EL1, Auxiliary ID Register, EL1* on page B2-304
AFSR0_EL1	3	c5	0	c1	0	32	*B2.8 AFSR0_EL1, Auxiliary Fault Status Register 0, EL1* on page B2-298
AFSR0_EL2	3	c5	4	c1	0	32	*B2.9 AFSR0_EL2, Auxiliary Fault Status Register 0, EL2* on page B2-299
AFSR0_EL3	3	c5	6	c1	0	32	*B2.10 AFSR0_EL3, Auxiliary Fault Status Register 0, EL3* on page B2-300
AFSR1_EL1	3	c5	0	c1	1	32	*B2.11 AFSR1_EL1, Auxiliary Fault Status Register 1, EL1* on page B2-301
AFSR1_EL2	3	c5	4	c1	1	32	*B2.12 AFSR1_EL2, Auxiliary Fault Status Register 1, EL2* on page B2-302
AFSR1_EL3	3	c5	6	c1	1	32	*B2.13 AFSR1_EL3, Auxiliary Fault Status Register 1, EL3* on page B2-303
AMAIR_EL1	3	c10	0	c3	0	64	*B2.15 AMAIR_EL1, Auxiliary Memory Attribute Indirection Register, EL1* on page B2-305
AMAIR_EL2	3	c10	4	c3	0	64	*B2.16 AMAIR_EL2, Auxiliary Memory Attribute Indirection Register, EL2* on page B2-306
AMAIR_EL3	3	c10	6	c3	0	64	*B2.17 AMAIR_EL3, Auxiliary Memory Attribute Indirection Register, EL3* on page B2-307
CCSIDR_EL1	3	c0	1	c0	0	32	*B2.18 CCSIDR_EL1, Cache Size ID Register, EL1* on page B2-308

Table B2-1 Registers with implementation defined bit fields (continued)

Name	Op0	CRn	Op1	CRm	Op2	Width	Description
CLIDR_EL1	3	c0	1	c0	1	64	*B2.19 CLIDR_EL1, Cache Level ID Register, EL1* on page B2-310
CPACR_EL1	3	c1	0	c0	2	32	*B2.20 CPACR_EL1, Architectural Feature Access Control Register, EL1* on page B2-312
CPTR_EL2	3	c1	4	c1	2	32	*B2.21 CPTR_EL2, Architectural Feature Trap Register, EL2* on page B2-313
CPTR_EL3	3	c1	6	c1	2	32	*B2.22 CPTR_EL3, Architectural Feature Trap Register, EL3* on page B2-314
CSSELR_EL1	3	c0	2	c0	0	32	*B2.32 CSSELR_EL1, Cache Size Selection Register, EL1* on page B2-334
CTR_EL0	3	c0	3	c0	1	32	*B2.33 CTR_EL0, Cache Type Register, EL0* on page B2-335
DISR_EL1	3	c12	0	c1	1	64	*B2.35 DISR_EL1, Deferred Interrupt Status Register, EL1* on page B2-338
ERRIDR_EL1	3	c5	0	c3	0	32	*B2.36 ERRIDR_EL1, Error ID Register, EL1* on page B2-339
ERRSELR_EL1	3	c5	0	c3	1	32	*B2.37 ERRSELR_EL1, Error Record Select Register, EL1* on page B2-340
ERXADDR_EL1	3	c5	0	c4	3	64	*B2.38 ERXADDR_EL1, Selected Error Record Address Register, EL1* on page B2-341
ERXCTLR_EL1	3	c5	0	c4	1	64	*B2.39 ERXCTLR_EL1, Selected Error Record Control Register, EL1* on page B2-342
ERXFR_EL1	3	c5	0	c4	0	64	*B2.40 ERXFR_EL1, Selected Error Record Feature Register, EL1* on page B2-343
ERXMISC0_EL1	3	c5	0	c5	0	64	*B2.41 ERXMISC0_EL1, Selected Error Record Miscellaneous Register 0, EL1* on page B2-344
ERXMISC1_EL1	3	c5	0	c5	1	64	*B2.42 ERXMISC1_EL1, Selected Error Record Miscellaneous Register 1, EL1* on page B2-345
ERXSTATUS_EL1	3	c5	0	c4	2	32	*B2.46 ERXSTATUS_EL1, Selected Error Record Primary Status Register, EL1* on page B2-351
ESR_EL1	3	c5	0	c2	0	32	*B2.47 ESR_EL1, Exception Syndrome Register, EL1* on page B2-352
ESR_EL2	3	c5	4	c2	0	32	*B2.48 ESR_EL2, Exception Syndrome Register, EL2* on page B2-353
ESR_EL3	3	c5	6	c2	0	32	*B2.49 ESR_EL3, Exception Syndrome Register, EL3* on page B2-354
HACR_EL2	3	c1	4	c1	7	32	*B2.50 HACR_EL2, Hyp Auxiliary Configuration Register, EL2* on page B2-355
HCR_EL2	3	c1	4	c1	0	64	*B2.51 HCR_EL2, Hypervisor Configuration Register, EL2* on page B2-356
ID_AFR0_EL1	3	c0	0	c1	3	32	*B2.62 ID_AFR0_EL1, AArch32 Auxiliary Feature Register 0, EL1* on page B2-373

Table B2-1 Registers with implementation defined bit fields (continued)

Name	Op0	CRn	Op1	CRm	Op2	Width	Description
ID_DFR0_EL1	3	c0	0	c1	2	32	*B2.63 ID_DFR0_EL1, AArch32 Debug Feature Register 0, EL1* on page B2-374
ID_ISAR0_EL1	3	c0	0	c2	0	32	*B2.64 ID_ISAR0_EL1, AArch32 Instruction Set Attribute Register 0, EL1* on page B2-376
ID_ISAR1_EL1	3	c0	0	c2	1	32	*B2.65 ID_ISAR1_EL1, AArch32 Instruction Set Attribute Register 1, EL1* on page B2-378
ID_ISAR2_EL1	3	c0	0	c2	2	32	*B2.66 ID_ISAR2_EL1, AArch32 Instruction Set Attribute Register 2, EL1* on page B2-380
ID_ISAR3_EL1	3	c0	0	c2	3	32	*B2.67 ID_ISAR3_EL1, AArch32 Instruction Set Attribute Register 3, EL1* on page B2-382
ID_ISAR4_EL1	3	c0	0	c2	4	32	*B2.68 ID_ISAR4_EL1, AArch32 Instruction Set Attribute Register 4, EL1* on page B2-384
ID_ISAR5_EL1	3	c0	0	c2	5	32	*B2.69 ID_ISAR5_EL1, AArch32 Instruction Set Attribute Register 5, EL1* on page B2-386
ID_ISAR6_EL1	3	c0	0	c2	7	32	*B2.70 ID_ISAR6_EL1, AArch32 Instruction Set Attribute Register 6, EL1* on page B2-388
ID_MMFR0_EL1	3	c0	0	c1	4	32	*B2.71 ID_MMFR0_EL1, AArch32 Memory Model Feature Register 0, EL1* on page B2-389
ID_MMFR1_EL1	3	c0	0	c1	5	32	*B2.72 ID_MMFR1_EL1, AArch32 Memory Model Feature Register 1, EL1* on page B2-391
ID_MMFR2_EL1	3	c0	0	c1	6	32	*B2.73 ID_MMFR2_EL1, AArch32 Memory Model Feature Register 2, EL1* on page B2-393
ID_MMFR3_EL1	3	c0	0	c1	7	32	*B2.74 ID_MMFR3_EL1, AArch32 Memory Model Feature Register 3, EL1* on page B2-395
ID_MMFR4_EL1	3	c0	0	c2	6	32	*B2.75 ID_MMFR4_EL1, AArch32 Memory Model Feature Register 4, EL1* on page B2-397
ID_PFR0_EL1	3	c0	0	c1	0	32	*B2.76 ID_PFR0_EL1, AArch32 Processor Feature Register 0, EL1* on page B2-399
ID_PFR1_EL1	3	c0	0	c1	1	32	*B2.77 ID_PFR1_EL1, AArch32 Core Feature Register 1, EL1* on page B2-400
ID_AA64DFR0_EL1	3	c0	0	c5	0	64	*B2.54 ID_AA64DFR0_EL1, AArch64 Debug Feature Register 0, EL1* on page B2-360
ID_AA64ISAR0_EL1	3	c0	0	c6	0	64	*B2.56 ID_AA64ISAR0_EL1, AArch64 Instruction Set Attribute Register 0, EL1* on page B2-363
ID_AA64ISAR1_EL1	3	c0	0	c6	1	64	*B2.57 ID_AA64ISAR1_EL1, AArch64 Instruction Set Attribute Register 1, EL1* on page B2-365
ID_AA64MMFR0_EL1	3	c0	0	c7	0	64	*B2.58 ID_AA64MMFR0_EL1, AArch64 Memory Model Feature Register 0, EL1* on page B2-366
ID_AA64MMFR1_EL1	3	c0	0	c7	1	64	*B2.59 ID_AA64MMFR1_EL1, AArch64 Memory Model Feature Register 1, EL1* on page B2-368
ID_AA64MMFR2_EL1	3	c0	0	c7	2	64	*B2.60 ID_AA64MMFR2_EL1, AArch64 Memory Model Feature Register 2, EL1* on page B2-370

Table B2-1 Registers with implementation defined bit fields (continued)

Name	Op0	CRn	Op1	CRm	Op2	Width	Description
ID_AA64PFR0_EL1	3	c0	0	c4	0	64	*B2.61 ID_AA64PFR0_EL1, AArch64 Processor Feature Register 0, EL1* on page B2-371
IFSR32_EL2	3	c5	4	c0	1	32	*B2.78 IFSR32_EL2, Instruction Fault Status Register, EL2* on page B2-401
LORC_EL1	3	c10	0	c4	3	64	*B2.79 LORC_EL1, LORegion Control Register, EL1* on page B2-403
LORID_EL1	3	c10	0	c4	7	64	*B2.80 LORID_EL1, LORegion ID Register, EL1* on page B2-404
LORN_EL1	3	c10	0	c4	2	64	*B2.81 LORN_EL1, LORegion Number Register, EL1* on page B2-405
MDCR_EL3	3	c1	6	c3	1	32	*B2.82 MDCR_EL3, Monitor Debug Configuration Register, EL3* on page B2-406
MIDR_EL1	3	c0	0	c0	0	32	*B2.83 MIDR_EL1, Main ID Register, EL1* on page B2-408
MPIDR_EL1	3	c0	0	c0	5	64	*B2.84 MPIDR_EL1, Multiprocessor Affinity Register, EL1* on page B2-409
PAR_EL1	3	c7	0	c4	0	64	*B2.85 PAR_EL1, Physical Address Register, EL1* on page B2-411
RVBAR_EL3	3	c12	6	c0	1	64	*B2.88 RVBAR_EL3, Reset Vector Base Address Register, EL3* on page B2-415
REVIDR_EL1	3	c0	0	c0	6	32	*B2.86 REVIDR_EL1, Revision ID Register, EL1* on page B2-412
SCTLR_EL1	3	c1	0	c0	0	32	*B2.89 SCTLR_EL1, System Control Register, EL1* on page B2-416
SCTLR_EL3	3	c1	6	c0	0	32	*B2.91 SCTLR_EL3, System Control Register, EL3* on page B2-419
TCR_EL1	3	c2	0	c0	2	64	*B2.92 TCR_EL1, Translation Control Register, EL1* on page B2-421
TCR_EL2	3	c2	4	c0	2	64	*B2.93 TCR_EL2, Translation Control Register, EL2* on page B2-422
TCR_EL3	3	c2	6	c0	2	64	*B2.94 TCR_EL3, Translation Control Register, EL3* on page B2-423
TTBR0_EL1	3	c2	0	c0	0	64	*B2.95 TTBR0_EL1, Translation Table Base Register 0, EL1* on page B2-424
TTBR0_EL2	3	c2	4	c0	0	64	*B2.96 TTBR0_EL2, Translation Table Base Register 0, EL2* on page B2-425
TTBR0_EL3	3	c2	6	c0	0	64	*B2.97 TTBR0_EL3, Translation Table Base Register 0, EL3* on page B2-426
TTBR1_EL1	3	c2	0	c0	1	64	*B2.98 TTBR1_EL1, Translation Table Base Register 1, EL1* on page B2-427
TTBR1_EL2	3	c2	4	c0	1	64	*B2.99 TTBR1_EL2, Translation Table Base Register 1, EL2* on page B2-428
VDISR_EL2	3	c12	4	c1	1	64	*B2.100 VDISR_EL2, Virtual Deferred Interrupt Status Register, EL2* on page B2-429

Table B2-1 Registers with implementation defined bit fields (continued)

Name	Op0	CRn	Op1	CRm	Op2	Width	Description
VSESR_EL2	3	c5	4	c2	3	64	*B2.101 VSESR_EL2, Virtual SError Exception Syndrome Register* on page B2-432
VTCR_EL2	3	c2	4	c1	2	32	*B2.102 VTCR_EL2, Virtualization Translation Control Register, EL2* on page B2-434
VTTBR_EL2	3	c2	4	c1	0	64	*B2.103 VTTBR_EL2, Virtualization Translation Table Base Register, EL2* on page B2-435

Table B2-2 Other architecturally defined registers

Name	Op0	CRn	Op1	CRm	Op2	Width	Description
AFSR0_EL12	3	c5	5	1	0	32	Auxiliary Fault Status Register 0
AFSR1_EL12	3	c5	5	1	1	32	Auxiliary Fault Status Register 1
AMAIR_EL12	3	c10	5	c3	0	64	Auxiliary Memory Attribute Indirection Register
CNTFRQ_EL0	3	c14	3	0	0	32	Counter-timer Frequency register
CNTHCTL_EL2	3	c14	4	c1	0	32	Counter-timer Hypervisor Control register
CNTHP_CTL_EL2	3	c14	4	c2	1	32	Counter-timer Hypervisor Physical Timer Control register
CNTHP_CVAL_EL2	3	c14	4	c2	2	64	Counter-timer Hyp Physical CompareValue register
CNTHP_TVAL_EL2	3	c14	4	c2	0	32	Counter-timer Hyp Physical Timer TimerValue register
CNTHV_CTL_EL2	3	c14	4	c3	1	32	Counter-timer Virtual Timer Control register
CNTHV_CVAL_EL2	3	c14	4	c3	2	64	Counter-timer Virtual Timer CompareValue register
CNTHV_TVAL_EL2	3	c14	4	c3	0	32	Counter-timer Virtual Timer TimerValue register
CNTKCTL_EL1	3	c14	0	c1	0	32	Counter-timer Kernel Control register
CNTKCTL_EL12	3	c14	5	c1	0	32	Counter-timer Kernel Control register
CNTP_CTL_EL0	3	c14	3	c2	1	32	Counter-timer Physical Timer Control register
CNTP_CTL_EL02	3	c14	5	c2	1	32	Counter-timer Physical Timer Control register
CNTP_CVAL_EL0	3	c14	3	c2	2	64	Counter-timer Physical Timer CompareValue register
CNTP_CVAL_EL02	3	c14	5	c2	2	64	Counter-timer Physical Timer CompareValue register
CNTP_TVAL_EL0	3	c14	3	c2	0	32	Counter-timer Physical Timer TimerValue register
CNTP_TVAL_EL02	3	c14	5	c2	0	32	Counter-timer Physical Timer TimerValue register
CNTPCT_EL0	3	c14	3	c0	1	64	Counter-timer Physical Count register
CNTPS_CTL_EL1	3	c14	7	c2	1	32	Counter-timer Physical Secure Timer Control register
CNTPS_CVAL_EL1	3	c14	7	c2	2	64	Counter-timer Physical Secure Timer CompareValue register
CNTPS_TVAL_EL1	3	c14	7	c2	0	32	Counter-timer Physical Secure Timer TimerValue register
CNTV_CTL_EL0	3	c14	3	c3	1	32	Counter-timer Virtual Timer Control register
CNTV_CTL_EL02	3	c14	5	c3	1	32	Counter-timer Virtual Timer Control register
CNTV_CVAL_EL0	3	c14	3	c3	2	64	Counter-timer Virtual Timer CompareValue register
CNTV_CVAL_EL02	3	c14	5	c3	2	64	Counter-timer Virtual Timer CompareValue register

Table B2-2 Other architecturally defined registers (continued)

Name	Op0	CRn	Op1	CRm	Op2	Width	Description
CNTV_TVAL_EL0	3	c14	3	c3	0	32	Counter-timer Virtual Timer TimerValue register
CNTV_TVAL_EL02	3	c14	5	c3	0	32	Counter-timer Virtual Timer TimerValue register
CNTVCT_EL0	3	c14	3	c0	2	64	Counter-timer Virtual Count register
CNTVOFF_EL2	3	c14	4	c0	3	64	Counter-timer Virtual Offset register
CONTEXTIDR_EL1	3	c13	0	c0	1	32	Context ID Register (EL1)
CONTEXTIDR_EL12	3	c13	5	c0	1	32	Context ID Register (EL12)
CONTEXTIDR_EL2	3	c13	4	c0	1	32	Context ID Register (EL2)
CPACR_EL12	3	c1	5	c0	2	32	Architectural Feature Access Control Register
CPTR_EL3	3	c1	6	c1	2	32	Architectural Feature Trap Register (EL3)
DACR32_EL2	3	c3	4	c0	0	32	Domain Access Control Register
ESR_EL12	3	c5	5	c2	0	32	Exception Syndrome Register (EL12)
FAR_EL1	3	c6	0	c0	0	64	Fault Address Register (EL1)
FAR_EL12	3	c6	5	c0	0	64	Fault Address Register (EL12)
FAR_EL2	3	c6	4	c0	0	64	Fault Address Register (EL2)
FAR_EL3	3	c6	6	c0	0	64	Fault Address Register (EL3)
FPEXC32_EL2	3	c5	4	c3	0	32	Floating-point Exception Control register
HPFAR_EL2	3	c6	4	c0	4	64	Hypervisor IPA Fault Address Register
HSTR_EL2	3	c1	4	c1	3	32	Hypervisor System Trap Register
ID_AA64AFR0_EL1	3	c0	0	c5	4	64	AArch64 Auxiliary Feature Register 0
ID_AA64AFR1_EL1	3	c0	0	c5	5	64	AArch64 Auxiliary Feature Register 1
ID_AA64DFR1_EL1	3	c0	0	c5	1	64	AArch64 Debug Feature Register 1
ID_AA64PFR1_EL1	3	c0	0	c4	1	64	AArch64 Core Feature Register 1
ISR_EL1	3	c12	0	c1	0	32	Interrupt Status Register
LOREA_EL1	3	c10	0	c4	1	64	LORegion End Address Register
LORSA_EL1	3	c10	0	c4	0	64	LORegion Start Address Register
MAIR_EL1	3	c10	0	c2	0	64	Memory Attribute Indirection Register (EL1)
MAIR_EL12	3	c10	5	c2	0	64	Memory Attribute Indirection Register (EL12)
MAIR_EL2	3	c10	4	c2	0	64	Memory Attribute Indirection Register (EL2)
MAIR_EL3	3	c10	6	c2	0	64	Memory Attribute Indirection Register (EL3)
MDCR_EL2	3	c1	4	c1	1	32	Monitor Debug Configuration Register
MVFR0_EL1	3	c0	0	c3	0	32	AArch32 Media and VFP Feature Register 0
MVFR1_EL1	3	c0	0	c3	1	32	AArch32 Media and VFP Feature Register 1
MVFR2_EL1	3	c0	0	c3	2	32	AArch32 Media and VFP Feature Register 2
RMR_EL3	3	c12	6	c0	2	32	Reset Management Register
SCR_EL3	3	c1	6	c1	0	32	Secure Configuration Register

Table B2-2 Other architecturally defined registers (continued)

Name	Op0	CRn	Op1	CRm	Op2	Width	Description
SCTLR_EL12	3	c1	5	c0	0	32	System Control Register (EL12)
SCTLR_EL2	3	c1	4	c0	0	32	System Control Register (EL2)
SDER32_EL3	3	c1	6	c1	1	32	AArch32 Secure Debug Enable Register
TCR_EL12	3	c2	5	c0	2	64	Translation Control Register (EL12)
TPIDR_EL0	3	c13	3	c0	2	64	EL0 Read/Write Software Thread ID Register
TPIDR_EL1	3	c13	0	c0	4	64	EL1 Software Thread ID Register
TPIDR_EL2	3	c13	4	c0	2	64	EL2 Software Thread ID Register
TPIDR_EL3	3	c13	6	c0	2	64	EL3 Software Thread ID Register
TPIDRRO_EL0	3	c13	3	c0	3	64	EL0 Read-Only Software Thread ID Register
TTBR0_EL12	3	c2	5	c0	0	64	Translation Table Base Register 0 (EL12)
TTBR1_EL12	3	c2	5	c0	1	64	Translation Table Base Register 1 (EL12)
VBAR_EL1	3	c12	0	c0	0	64	Vector Base Address Register (EL1)
VBAR_EL12	3	c12	5	c0	0	64	Vector Base Address Register (EL12)
VBAR_EL2	3	c12	4	c0	0	64	Vector Base Address Register (EL2)
VBAR_EL3	3	c12	6	c0	0	64	Vector Base Address Register (EL3)
VMPIDR_EL2	3	c0	4	c0	5	64	Virtualization Multiprocessor ID Register
VPIDR_EL2	3	c0	4	c0	0	32	Virtualization Core ID Register

B2.3 AArch64 implementation defined register summary

This section describes the AArch64 registers in the core that are implementation defined.

The following tables lists the AArch 64 implementation defined registers, sorted by opcode.

Table B2-3 AArch64 implementation defined registers

Name	Op0	CRn	Op1	CRm	Op2	Width	Description
CPUACTLR_EL1	3	c15	0	c1	0	64	*B2.23 CPUACTLR_EL1, CPU Auxiliary Control Register, EL1* on page B2-315
CPUACTLR2_EL1	3	c15	0	c1	1	64	*B2.24 CPUACTLR2_EL1, CPU Auxiliary Control Register 2, EL1* on page B2-317
CPUCFR_EL1	3	c15	0	c0	0	32	*B2.25 CPUCFR_EL1, CPU Configuration Register, EL1* on page B2-319
CPUECTLR_EL1	3	c15	0	c1	4	64	*B2.26 CPUECTLR_EL1, CPU Extended Control Register, EL1* on page B2-321
CPUPCR_EL3	3	15	6	c8	1	64	*B2.27 CPUPCR_EL3, CPU Private Control Register, EL3* on page B2-324
CPUPMR_EL3	3	c15	6	c8	3	64	*B2.28 CPUPMR_EL3, CPU Private Mask Register, EL3* on page B2-326
CPUPOR_EL3	3	c15	6	c8	2	64	*B2.29 CPUPOR_EL3, CPU Private Operation Register, EL3* on page B2-328
CPUPSELR_EL3	3	c15	6	c8	0	32	*B2.30 CPUPSELR_EL3, CPU Private Selection Register, EL3* on page B2-330
CPUPWRCTLR_EL1	3	c15	0	c2	7	32	*B2.31 CPUPWRCTLR_EL1, Power Control Register, EL1* on page B2-332
ERXPFGCDNR_EL1	3	c15	0	c2	2	32	*B2.43 ERXPFGCDNR_EL1, Selected Error Pseudo Fault Generation Count Down Register, EL1* on page B2-346
ERXPFGCTLR_EL1	3	c15	0	c2	1	32	*B2.44 ERXPFGCTLR_EL1, Selected Error Pseudo Fault Generation Control Register, EL1* on page B2-348
ERXPFGFR_EL1	3	c15	0	c2	0	32	*B2.45 ERXPFGFR_EL1, Selected Pseudo Fault Generation Feature Register, EL1* on page B2-350

The following table shows the 32-bit wide implementation defined Cluster registers. Details of these registers can be found in *Arm® DynamIQ™ Shared Unit Technical Reference Manual*

Table B2-4 Cluster registers

Name	Op0	CRn	op1	CRm	op2	Width	Description
CLUSTERCFR_EL1	3	c15	0	c3	0	32-bit	Cluster configuration register.
CLUSTERIDR_EL1	3	c15	0	c3	1	32-bit	Cluster main revision ID.
CLUSTEREVIDR_EL1	3	c15	0	c3	2	32-bit	Cluster ECO ID.
CLUSTERACTLR_EL1	3	c15	0	c3	3	32-bit	Cluster auxiliary control register.
CLUSTERECTLR_EL1	3	c15	0	c3	4	32-bit	Cluster extended control register.
CLUSTERPWRCTLR_EL1	3	c15	0	c3	5	32-bit	Cluster power control register.
CLUSTERPWRDN_EL1	3	c15	0	c3	6	32-bit	Cluster power down register.
CLUSTERPWRSTAT_EL1	3	c15	0	c3	7	32-bit	Cluster power status register.

Table B2-4 Cluster registers (continued)

Name	Op0	CRn	op1	CRm	op2	Width	Description
CLUSTERTHREADSID_EL1	3	c15	0	c4	0	32-bit	Cluster thread scheme ID register.
CLUSTERACPSID_EL1	3	c15	0	c4	1	32-bit	Cluster ACP scheme ID register.
CLUSTERSTASHSID_EL1	3	c15	0	c4	2	32-bit	Cluster stash scheme ID register.
CLUSTERPARTCR_EL1	3	c15	0	c4	3	32-bit	Cluster partition control register.
CLUSTERBUSQOS_EL1	3	c15	0	c4	4	32-bit	Cluster bus QoS control register.
CLUSTERL3HIT_EL1	3	c15	0	c4	5	32-bit	Cluster L3 hit counter register.
CLUSTERL3MISS_EL1	3	c15	0	c4	6	32-bit	Cluster L3 miss counter register.
CLUSTERTHREADSIDOVR_EL1	3	c15	0	c4	7	32-bit	Cluster thread scheme ID override register.
CLUSTERPM*_ELx	3	c15	0 or 6	c5-c6	0-7	32-bit	Cluster PMU registers

B2.4 AArch64 registers by functional group

This section identifies the AArch64 registers by their functional groups and applies to the registers in the core that are implementation defined or have micro-architectural bit fields.

Reset values are provided for these registers.

Identification registers

Name	Type	Reset	Description
AIDR_EL1	RO	`0x00000000`	*B2.14 AIDR_EL1, Auxiliary ID Register, EL1* on page B2-304
CCSIDR__EL1	RO	-	*B2.18 CCSIDR_EL1, Cache Size ID Register, EL1* on page B2-308
CLIDR_EL1	RO	• `0xC3000123` if L3 cache present. • `0x82000023` if no L3 cache.	*B2.19 CLIDR_EL1, Cache Level ID Register, EL1* on page B2-310
CSSELR_EL1	RW	UNK	*B2.32 CSSELR_EL1, Cache Size Selection Register, EL1* on page B2-334
CTR_EL0	RO	`0x84448004`	*B2.33 CTR_EL0, Cache Type Register, EL0* on page B2-335
DCZID_EL0	RO	`0x00000004`	*B2.34 DCZID_EL0, Data Cache Zero ID Register, EL0* on page B2-337
ERRIDR_EL1	RO	-	*B2.36 ERRIDR_EL1, Error ID Register, EL1* on page B2-339
ID_AA64AFR0_EL1	RO	`0x00000000`	*B2.52 ID_AA64AFR0_EL1, AArch64 Auxiliary Feature Register 0* on page B2-358
ID_AA64AFR1_EL1	RO	`0x00000000`	*B2.53 ID_AA64AFR1_EL1, AArch64 Auxiliary Feature Register 1* on page B2-359
ID_AA64DFR0_EL1	RO	`0x0000000010305408`	*B2.54 ID_AA64DFR0_EL1, AArch64 Debug Feature Register 0, EL1* on page B2-360
ID_AA64DFR1_EL1	RO	`0x00000000`	*B2.55 ID_AA64DFR1_EL1, AArch64 Debug Feature Register 1, EL1* on page B2-362
ID_AA64ISAR0_EL1	RO	• `0x0000000010211120` if the Cryptographic Extension is implemented. • `0x0000000010210000` if the Cryptographic Extension is not implemented.	*B2.56 ID_AA64ISAR0_EL1, AArch64 Instruction Set Attribute Register 0, EL1* on page B2-363
ID_AA64ISAR1_EL1	RO	`0x0000000000100001`	*B2.57 ID_AA64ISAR1_EL1, AArch64 Instruction Set Attribute Register 1, EL1* on page B2-365
ID_AA64MMFR0_EL1	RO	`0x0000000000101124`	*B2.58 ID_AA64MMFR0_EL1, AArch64 Memory Model Feature Register 0, EL1* on page B2-366
ID_AA64MMFR1_EL1	RO	`0x0000000010212122`	*B2.59 ID_AA64MMFR1_EL1, AArch64 Memory Model Feature Register 1, EL1* on page B2-368
ID_AA64MMFR2_EL1	RO	`0x0000000000001011`	*B2.60 ID_AA64MMFR2_EL1, AArch64 Memory Model Feature Register 2, EL1* on page B2-370

(continued)

Name	Type	Reset	Description
ID_AA64PFR0_EL1	RO	• `0x0000000010112222` if the GICv4 interface is disabled. • `0x0000000011112222` if the GICv4 interface is enabled.	*B2.61 ID_AA64PFR0_EL1, AArch64 Processor Feature Register 0, EL1* on page B2-371
ID_AFR0_EL1	RO	`0x00000000`	*B2.62 ID_AFR0_EL1, AArch32 Auxiliary Feature Register 0, EL1* on page B2-373
ID_DFR0_EL1	RO	`0x04010088`	*B2.63 ID_DFR0_EL1, AArch32 Debug Feature Register 0, EL1* on page B2-374 Bits [19:16] are `0x1` if ETM is implemented, and `0x0` otherwise.
ID_ISAR0_EL1	RO	`0x02101110`	*B2.64 ID_ISAR0_EL1, AArch32 Instruction Set Attribute Register 0, EL1* on page B2-376
ID_ISAR1_EL1	RO	`0x13112111`	*B2.65 ID_ISAR1_EL1, AArch32 Instruction Set Attribute Register 1, EL1* on page B2-378
ID_ISAR2_EL1	RO	`0x21232042`	*B2.66 ID_ISAR2_EL1, AArch32 Instruction Set Attribute Register 2, EL1* on page B2-380
ID_ISAR3_EL1	RO	`0x01112131`	*B2.67 ID_ISAR3_EL1, AArch32 Instruction Set Attribute Register 3, EL1* on page B2-382
ID_ISAR4_EL1	RO	`0x00011142`	*B2.68 ID_ISAR4_EL1, AArch32 Instruction Set Attribute Register 4, EL1* on page B2-384
ID_ISAR5_EL1	RO	`0x00011121`	*B2.69 ID_ISAR5_EL1, AArch32 Instruction Set Attribute Register 5, EL1* on page B2-386 ID_ISAR5 has the value `0x00010001` if the Cryptographic Extension is not implemented and enabled.
ID_ISAR6_EL1	RO	`0x00000010`	*B2.70 ID_ISAR6_EL1, AArch32 Instruction Set Attribute Register 6, EL1* on page B2-388
ID_MMFR0_EL1	RO	`0x10201105`	*B2.71 ID_MMFR0_EL1, AArch32 Memory Model Feature Register 0, EL1* on page B2-389
ID_MMFR1_EL1	RO	`0x40000000`	*B2.72 ID_MMFR1_EL1, AArch32 Memory Model Feature Register 1, EL1* on page B2-391
ID_MMFR2_EL1	RO	`0x01260000`	*B2.73 ID_MMFR2_EL1, AArch32 Memory Model Feature Register 2, EL1* on page B2-393
ID_MMFR3_EL1	RO	`0x02122211`	*B2.74 ID_MMFR3_EL1, AArch32 Memory Model Feature Register 3, EL1* on page B2-395
ID_MMFR4_EL1	RO	`0x00021110`	*B2.75 ID_MMFR4_EL1, AArch32 Memory Model Feature Register 4, EL1* on page B2-397
ID_PFR0_EL1	RO	`0x00000131`	*B2.76 ID_PFR0_EL1, AArch32 Processor Feature Register 0, EL1* on page B2-399
ID_PFR1_EL1	RO	`0x10011011`	*B2.77 ID_PFR1_EL1, AArch32 Core Feature Register 1, EL1* on page B2-400 Bits [31:28] are `0x1` if the GIC CPU interface is implemented and enabled, and `0x0` otherwise.

(continued)

Name	Type	Reset	Description
LORID_EL1	RO	0x0000000000040004	*B2.80 LORID_EL1, LORegion ID Register, EL1* on page B2-404
MIDR_EL1	RO	0x412FD0A1	*B2.83 MIDR_EL1, Main ID Register, EL1* on page B2-408
MPIDR_EL1	RO	-	*B2.84 MPIDR_EL1, Multiprocessor Affinity Register, EL1* on page B2-409
REVIDR_EL1	RO	0x00000000	*B2.86 REVIDR_EL1, Revision ID Register, EL1* on page B2-412
VMPIDR_EL2	RW	-	Virtualization Multiprocessor ID Register EL2 The reset value is the value of MPIDR_EL1.
VPIDR_EL2	RW	-	Virtualization Core ID Register EL2 The reset value is the value of MIDR_EL1.

Other system control registers

Name	Type	Description
ACTLR_EL1	RW	*B2.5 ACTLR_EL1, Auxiliary Control Register, EL1* on page B2-293
ACTLR_EL2	RW	*B2.6 ACTLR_EL2, Auxiliary Control Register, EL2* on page B2-294
ACTLR_EL3	RW	*B2.7 ACTLR_EL3, Auxiliary Control Register, EL3* on page B2-296
CPACR_EL1	RW	*B2.20 CPACR_EL1, Architectural Feature Access Control Register, EL1* on page B2-312
SCTLR_EL1	RW	*B2.89 SCTLR_EL1, System Control Register, EL1* on page B2-416
SCTLR_EL3	RW	*B2.91 SCTLR_EL3, System Control Register, EL3* on page B2-419

Reliability, Availability, Serviceability (RAS) registers

Name	Type	Description
DISR_EL1	RW	*B2.35 DISR_EL1, Deferred Interrupt Status Register, EL1* on page B2-338
ERRIDR_EL1	RW	*B2.36 ERRIDR_EL1, Error ID Register, EL1* on page B2-339
ERRSELR_EL1	RW	*B2.37 ERRSELR_EL1, Error Record Select Register, EL1* on page B2-340
ERXADDR_EL1	RW	*B2.38 ERXADDR_EL1, Selected Error Record Address Register, EL1* on page B2-341
ERXCTLR_EL1	RW	*B2.39 ERXCTLR_EL1, Selected Error Record Control Register, EL1* on page B2-342
ERXFR_EL1	RO	*B2.40 ERXFR_EL1, Selected Error Record Feature Register, EL1* on page B2-343
ERXMISC0_EL1	RW	*B2.41 ERXMISC0_EL1, Selected Error Record Miscellaneous Register 0, EL1* on page B2-344
ERXMISC1_EL1	RW	*B2.42 ERXMISC1_EL1, Selected Error Record Miscellaneous Register 1, EL1* on page B2-345
ERXSTATUS_EL1	RW	*B2.46 ERXSTATUS_EL1, Selected Error Record Primary Status Register, EL1* on page B2-351
ERXPFGCDNR_EL1	RW	*B2.43 ERXPFGCDNR_EL1, Selected Error Pseudo Fault Generation Count Down Register, EL1* on page B2-346
ERXPFCTLR_EL1	RW	*B2.44 ERXPFGCTLR_EL1, Selected Error Pseudo Fault Generation Control Register, EL1* on page B2-348
ERXPFGFR_EL1	RO	*B2.45 ERXPFGFR_EL1, Selected Pseudo Fault Generation Feature Register, EL1* on page B2-350
HCR_EL2	RW	*B2.51 HCR_EL2, Hypervisor Configuration Register, EL2* on page B2-356

(continued)

Name	Type	Description
VDISR_EL2	RW	*B2.100 VDISR_EL2, Virtual Deferred Interrupt Status Register, EL2* on page B2-429
VSESR_EL2	RW	*B2.101 VSESR_EL2, Virtual SError Exception Syndrome Register* on page B2-432

Virtual Memory control registers

Name	Type	Description
AMAIR_EL1	RW	*B2.15 AMAIR_EL1, Auxiliary Memory Attribute Indirection Register, EL1* on page B2-305
AMAIR_EL2	RW	*B2.16 AMAIR_EL2, Auxiliary Memory Attribute Indirection Register, EL2* on page B2-306
AMAIR_EL3	RW	*B2.17 AMAIR_EL3, Auxiliary Memory Attribute Indirection Register, EL3* on page B2-307
LORC_EL1	RW	*B2.79 LORC_EL1, LORegion Control Register, EL1* on page B2-403
LOREA_EL1	RW	LORegion End Address Register EL1
LORID_EL1	RO	*B2.80 LORID_EL1, LORegion ID Register, EL1* on page B2-404
LORN_EL1	RW	*B2.81 LORN_EL1, LORegion Number Register, EL1* on page B2-405
LORSA_EL1	RW	LORegion Start Address Register EL1
TCR_EL1	RW	*B2.92 TCR_EL1, Translation Control Register, EL1* on page B2-421
TCR_EL2	RW	*B2.93 TCR_EL2, Translation Control Register, EL2* on page B2-422
TCR_EL3	RW	*B2.94 TCR_EL3, Translation Control Register, EL3* on page B2-423
TTBR0_EL1	RW	*B2.95 TTBR0_EL1, Translation Table Base Register 0, EL1* on page B2-424
TTBR0_EL2	RW	*B2.96 TTBR0_EL2, Translation Table Base Register 0, EL2* on page B2-425
TTBR0_EL3	RW	*B2.97 TTBR0_EL3, Translation Table Base Register 0, EL3* on page B2-426
TTBR1_EL1	RW	*B2.98 TTBR1_EL1, Translation Table Base Register 1, EL1* on page B2-427
TTBR1_EL2	RW	*B2.99 TTBR1_EL2, Translation Table Base Register 1, EL2* on page B2-428
VTTBR_EL2	RW	*B2.103 VTTBR_EL2, Virtualization Translation Table Base Register, EL2* on page B2-435

Virtualization registers

Name	Type	Description
ACTLR_EL2	RW	*B2.6 ACTLR_EL2, Auxiliary Control Register, EL2* on page B2-294
AFSR0_EL2	RW	*B2.9 AFSR0_EL2, Auxiliary Fault Status Register 0, EL2* on page B2-299
AFSR1_EL2	RW	*B2.12 AFSR1_EL2, Auxiliary Fault Status Register 1, EL2* on page B2-302
AMAIR_EL2	RW	*B2.16 AMAIR_EL2, Auxiliary Memory Attribute Indirection Register, EL2* on page B2-306
CPTR_EL2	RW	*B2.21 CPTR_EL2, Architectural Feature Trap Register, EL2* on page B2-313
ESR_EL2	RW	*B2.48 ESR_EL2, Exception Syndrome Register, EL2* on page B2-353
HACR_EL2	RW	*B2.50 HACR_EL2, Hyp Auxiliary Configuration Register, EL2* on page B2-355
HCR_EL2	RW	*B2.51 HCR_EL2, Hypervisor Configuration Register, EL2* on page B2-356
HPFAR_EL2	RW	Hypervisor IPA Fault Address Register EL2
TCR_EL2	RW	*B2.93 TCR_EL2, Translation Control Register, EL2* on page B2-422
VMPIDR_EL2	RW	Virtualization Multiprocessor ID Register EL2

(continued)

Name	Type	Description
VPIDR_EL2	RW	Virtualization Core ID Register EL2
VSESR_EL2	RW	*B2.101 VSESR_EL2, Virtual SError Exception Syndrome Register* on page B2-432
VTCR_EL2	RW	*B2.102 VTCR_EL2, Virtualization Translation Control Register, EL2* on page B2-434
VTTBR_EL2	RW	*B2.103 VTTBR_EL2, Virtualization Translation Table Base Register, EL2* on page B2-435

Exception and fault handling registers

Name	Type	Description
AFSR0_EL1	RW	*B2.8 AFSR0_EL1, Auxiliary Fault Status Register 0, EL1* on page B2-298
AFSR0_EL2	RW	*B2.9 AFSR0_EL2, Auxiliary Fault Status Register 0, EL2* on page B2-299
AFSR0_EL3	RW	*B2.10 AFSR0_EL3, Auxiliary Fault Status Register 0, EL3* on page B2-300
AFSR1_EL1	RW	*B2.11 AFSR1_EL1, Auxiliary Fault Status Register 1, EL1* on page B2-301
AFSR1_EL2	RW	*B2.12 AFSR1_EL2, Auxiliary Fault Status Register 1, EL2* on page B2-302
AFSR1_EL3	RW	*B2.13 AFSR1_EL3, Auxiliary Fault Status Register 1, EL3* on page B2-303
DISR_EL1	RW	*B2.35 DISR_EL1, Deferred Interrupt Status Register, EL1* on page B2-338
ESR_EL1	RW	*B2.47 ESR_EL1, Exception Syndrome Register, EL1* on page B2-352
ESR_EL2	RW	*B2.48 ESR_EL2, Exception Syndrome Register, EL2* on page B2-353
ESR_EL3	RW	*B2.49 ESR_EL3, Exception Syndrome Register, EL3* on page B2-354
HPFAR_EL2	RW	Hypervisor IPA Fault Address Register EL2
IFSR32_EL2	RW	*B2.78 IFSR32_EL2, Instruction Fault Status Register, EL2* on page B2-401
VDISR_EL2	RW	*B2.100 VDISR_EL2, Virtual Deferred Interrupt Status Register, EL2* on page B2-429
VSESR_EL2	RW	*B2.101 VSESR_EL2, Virtual SError Exception Syndrome Register* on page B2-432

Implementation defined registers

Name	Type	Description
CPUACTLR_EL1	RW	*B2.23 CPUACTLR_EL1, CPU Auxiliary Control Register, EL1* on page B2-315
CPUACTLR2_EL1	RW	*B2.24 CPUACTLR2_EL1, CPU Auxiliary Control Register 2, EL1* on page B2-317
CPUCFR_EL1	RO	*B2.25 CPUCFR_EL1, CPU Configuration Register, EL1* on page B2-319
CPUECTLR_EL1	RW	*B2.26 CPUECTLR_EL1, CPU Extended Control Register, EL1* on page B2-321
CPUPCR_EL3	RW	*B2.27 CPUPCR_EL3, CPU Private Control Register, EL3* on page B2-324
CPUPMR_EL3	RW	*B2.28 CPUPMR_EL3, CPU Private Mask Register, EL3* on page B2-326
CPUPOR_EL3	RW	*B2.29 CPUPOR_EL3, CPU Private Operation Register, EL3* on page B2-328
CPUPSELR_EL3	RW	*B2.30 CPUPSELR_EL3, CPU Private Selection Register, EL3* on page B2-330
CPUPWRCTLR_EL1	RW	*B2.31 CPUPWRCTLR_EL1, Power Control Register, EL1* on page B2-332
ERXPFGCDNR_EL1	RW	*B2.43 ERXPFGCDNR_EL1, Selected Error Pseudo Fault Generation Count Down Register, EL1* on page B2-346

(continued)

Name	Type	Description
ERXPFGCTLR_EL1	RW	*B2.44 ERXPFGCTLR_EL1, Selected Error Pseudo Fault Generation Control Register, EL1* on page B2-348
ERXPFGFR_EL1	RO	*B2.45 ERXPFGFR_EL1, Selected Pseudo Fault Generation Feature Register, EL1* on page B2-350

The following table shows the 32-bit wide implementation defined Cluster registers. Details of these registers can be found in *Arm® DynamIQ™ Shared Unit Technical Reference Manual*

Table B2-5 Cluster registers

Name	Copro	CRn	Opc1	CRm	Opc2	Width	Description
CLUSTERCFR_EL1	3	c15	0	c3	0	32-bit	Cluster configuration register.
CLUSTERIDR_EL1	3	c15	0	c3	1	32-bit	Cluster main revision ID.
CLUSTEREVIDR_EL1	3	c15	0	c3	2	32-bit	Cluster ECO ID.
CLUSTERACTLR_EL1	3	c15	0	c3	3	32-bit	Cluster auxiliary control register.
CLUSTERECTLR_EL1	3	c15	0	c3	4	32-bit	Cluster extended control register.
CLUSTERPWRCTLR_EL1	3	c15	0	c3	5	32-bit	Cluster power control register.
CLUSTERPWRDN_EL1	3	c15	0	c3	6	32-bit	Cluster power down register.
CLUSTERPWRSTAT_EL1	3	c15	0	c3	7	32-bit	Cluster power status register.
CLUSTERTHREADSID_EL1	3	c15	0	c4	0	32-bit	Cluster thread scheme ID register.
CLUSTERACPSID_EL1	3	c15	0	c4	1	32-bit	Cluster ACP scheme ID register.
CLUSTERSTASHSID_EL1	3	c15	0	c4	2	32-bit	Cluster stash scheme ID register.
CLUSTERPARTCR_EL1	3	c15	0	c4	3	32-bit	Cluster partition control register.
CLUSTERBUSQOS_EL1	3	c15	0	c4	4	32-bit	Cluster bus QoS control register.
CLUSTERL3HIT_EL1	3	c15	0	c4	5	32-bit	Cluster L3 hit counter register.
CLUSTERL3MISS_EL1	3	c15	0	c4	6	32-bit	Cluster L3 miss counter register.
CLUSTERPM*_ELx	3	c15	0 or 6	c5-c6	0-7	32-bit	Cluster PMU registers

Security

Name	Type	Description
ACTLR_EL3	RW	*B2.7 ACTLR_EL3, Auxiliary Control Register, EL3* on page B2-296
AFSR0_EL3	RW	*B2.10 AFSR0_EL3, Auxiliary Fault Status Register 0, EL3* on page B2-300
AFSR1_EL3	RW	*B2.13 AFSR1_EL3, Auxiliary Fault Status Register 1, EL3* on page B2-303
AMAIR_EL3	RW	*B2.17 AMAIR_EL3, Auxiliary Memory Attribute Indirection Register, EL3* on page B2-307
CPTR_EL3	RW	*B2.22 CPTR_EL3, Architectural Feature Trap Register, EL3* on page B2-314
MDCR_EL3	RW	*B2.82 MDCR_EL3, Monitor Debug Configuration Register, EL3* on page B2-406

Reset management registers

Name	Type	Description
RMR_EL3	RW	*B2.87 RMR_EL3, Reset Management Register* on page B2-413
RVBAR_EL3	RW	*B2.88 RVBAR_EL3, Reset Vector Base Address Register, EL3* on page B2-415

Address registers

Name	Type	Description
PAR_EL1	RW	*B2.85 PAR_EL1, Physical Address Register, EL1* on page B2-411

B2.5 ACTLR_EL1, Auxiliary Control Register, EL1

ACTLR_EL1 provides IMPLEMENTATION DEFINED configuration and control options for execution at EL1 and EL0. This register is not used in the A75 core.

ACTLR_EL1 is a 64-bit register, and is part of:

- The Other system control registers functional group.
- The IMPLEMENTATION DEFINED functional group.

Bit field descriptions

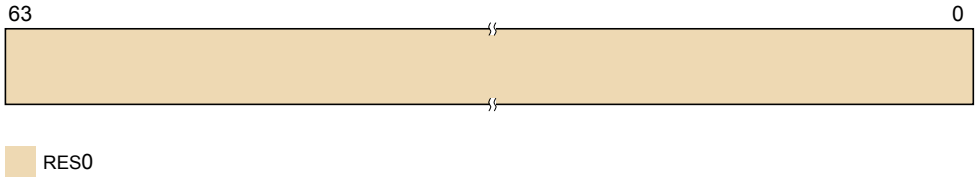

Figure B2-1 ACTLR_EL1 bit assignments

RES0, [63:0]

RES0 Reserved.

Configurations

AArch64 System register ACTLR_EL1 bits [31:0] are architecturally mapped to AArch32 System register ACTLR(NS). See *B1.5 ACTLR, Auxiliary Control Register* on page B1-136.

AArch64 System register ACTLR_EL1 bits [63:32] are architecturally mapped to AArch32 System register ACTLR2(S). See *B1.6 ACTLR2, Auxiliary Control Register 2* on page B1-138.

Bit fields and details that are not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

B2.6 ACTLR_EL2, Auxiliary Control Register, EL2

The ACTLR_EL2 provides IMPLEMENTATION DEFINED configuration and control options for EL2.

Bit field descriptions

ACTLR_EL2 is a 64-bit register, and is part of:

- The Virtualization registers functional group.
- The Other system control registers functional group.
- The IMPLEMENTATION DEFINED functional group.

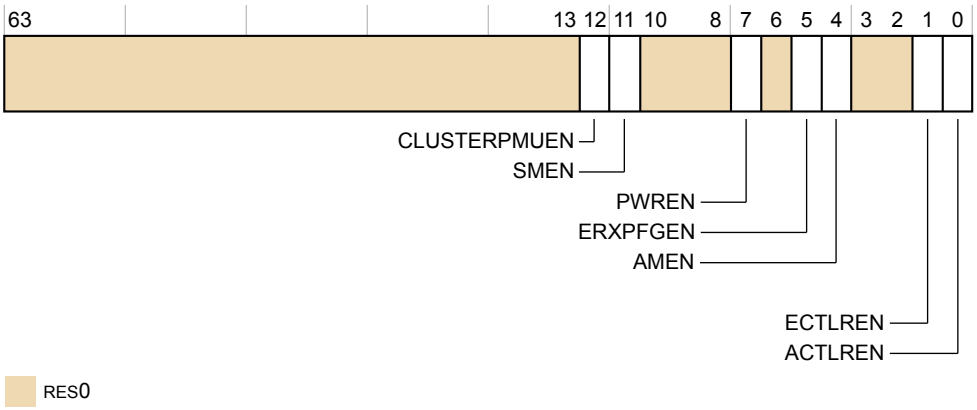

Figure B2-2 ACTLR_EL2 bit assignments

RES0, [63:13]

RES0 Reserved.

CLUSTERPMUEN, [12]

Performance Management Registers enable. The possible values are:

0 CLUSTERPM* registers are not write-accessible from a lower Exception level. This is the reset value.

1 CLUSTERPM* registers are write-accessible from EL1 Non-secure if they are write-accessible from EL2.

SMEN, [11]

Scheme Management Registers enable. The possible values are:

0 Registers CLUSTERACPSID, CLUSTERSTASHSID, CLUSTERPARTCR, CLUSTERBUSQOS, and CLUSTERTHREADSIDOVR are not write-accessible from EL1 Non-secure. This is the reset value.

1 Registers CLUSTERACPSID, CLUSTERSTASHSID, CLUSTERPARTCR, CLUSTERBUSQOS, and CLUSTERTHREADSIDOVR are write-accessible from EL1 Non-secure if they are write-accessible from EL2.

TSIDEN, [10]

Thread Scheme ID Register enable. The possible values are:

0 Register CLUSTERTHREADSID is not write-accessible from EL1 Non-secure. This is the reset value.

1 Register CLUSTERTHREADSID is write-accessible from EL1 Non-secure if they are write-accessible from EL2.

RES0, [9:8]

RES0 Reserved.

PWREN, [7]

Power Control Registers enable. The possible values are:

0 Registers CPUPWRCTLR, CLUSTERPWRCTLR, CLUSTERPWRDN, CLUSTERPWRSTAT, CLUSTERL3HIT and CLUSTERL3MISS are not write-accessible from EL1 Non-secure. This is the reset value.

1 Registers CPUPWRCTLR, CLUSTERPWRCTLR, CLUSTERPWRDN, CLUSTERPWRSTAT, CLUSTERL3HIT and CLUSTERL3MISS are write-accessible from EL1 Non-secure if they are write-accessible from EL2.

RES0, [6]

RES0 Reserved.

ERXPFGEN, [5]

Error Record Registers enable. The possible values are:

0 ERXPFG* are not write-accessible from EL1 Non-secure. This is the reset value.

1 ERXPFG* are write-accessible from EL1 Non-secure if they are write-accessible from EL2.

AMEN, [4]

Activity Monitor enable. The possible values are:

0 Non-secure accesses from EL1 and EL0 to activity monitor registers are trapped to EL2.

1 Non-secure accesses from EL1 and EL0 to activity monitor registers are not trapped to EL2.

RES0, [3:2]

RES0 Reserved.

ECTLREN, [1]

Extended Control Registers enable. The possible values are:

0 CPUECTLR and CLUSTERECTLR are not write-accessible from EL1 Non-secure. This is the reset value.

1 CPUECTLR and CLUSTERECTLR are write-accessible from EL1 Non-secure if they are write-accessible from EL2.

ACTLREN, [0]

Auxiliary Control Registers enable. The possible values are:

0 CPUACTLR, CPUACTLR2, and CLUSTERACTLR are not write-accessible from EL1 Non-secure. This is the reset value.

1 CPUACTLR, CPUACTLR2, and CLUSTERACTLR are write-accessible from EL1 Non-secure if they are write-accessible from EL2.

Configurations

ACTLR_EL2 bits [31:0] are architecturally mapped to the AArch32 HACTLR register. See *B1.46 HACTLR, Hyp Auxiliary Control Register* on page B1-196.

ACTLR_EL2 bits [63:32] are architecturally mapped to the AArch32 HACTLR2 register. See *B1.47 HACTLR2, Hyp Auxiliary Control Register 2* on page B1-198.

Bit fields and details that are not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

B2.7 ACTLR_EL3, Auxiliary Control Register, EL3

The ACTLR_EL3 provides IMPLEMENTATION DEFINED configuration and control options for EL3.

Bit field descriptions

ACTLR_EL3 is a 64-bit register, and is part of:

- The Other system control registers functional group.
- The Security registers functional group.
- The IMPLEMENTATION DEFINED functional group.

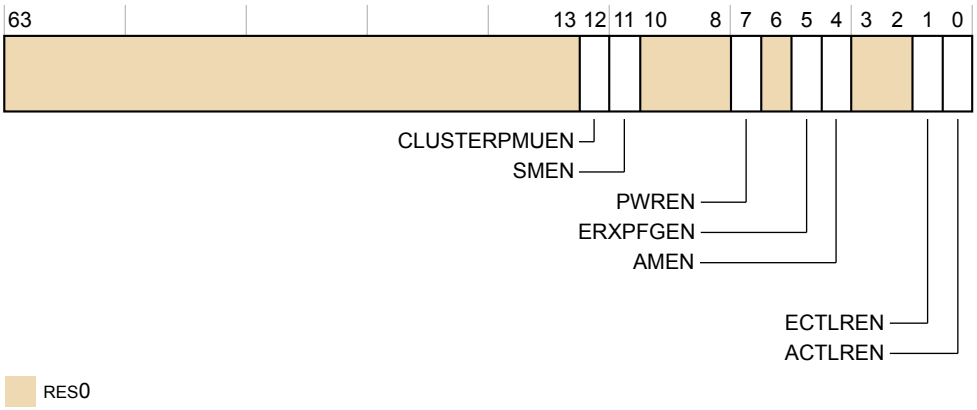

Figure B2-3 ACTLR_EL3 bit assignments

RES0, [63:13]

RES0 Reserved.

CLUSTERPMUEN, [12]

Performance Management Registers enable. The possible values are:

0 CLUSTERPM* registers are not write-accessible from a lower Exception level. This is the reset value.

1 CLUSTERPM* registers are write-accessible from EL2 and EL1 Secure.

SMEN, [11]

Scheme Management Registers enable. The possible values are:

0 Registers CLUSTERACPSID, CLUSTERSTASHSID, CLUSTERPARTCR, CLUSTERBUSQOS, and CLUSTERTHREADSIDOVR are not write-accessible from EL2 and EL1 Secure. This is the reset value.

1 Registers CLUSTERACPSID, CLUSTERSTASHSID, CLUSTERPARTCR, CLUSTERBUSQOS, and CLUSTERTHREADSIDOVR are write-accessible from EL2 and EL1 Secure.

TSIDEN, [10]

Thread Scheme ID Register enable. The possible values are:

0 Register CLUSTERTHREADSID is not write-accessible from EL2 and EL1 Secure. This is the reset value.

1 Register CLUSTERTHREADSID is write-accessible from EL2 and EL1 Secure.

RES0, [9:8]

RES0 Reserved.

PWREN, [7]

Power Control Registers enable. The possible values are:

0 Registers CPUPWRCTLR, CLUSTERPWRCTLR, CLUSTERPWRDN, CLUSTERPWRSTAT, CLUSTERL3HIT and CLUSTERL3MISS are not write-accessible from EL2 and EL1 Secure. This is the reset value.

1 Registers CPUPWRCTLR, CLUSTERPWRCTLR, CLUSTERPWRDN, CLUSTERPWRSTAT, CLUSTERL3HIT and CLUSTERL3MISS are write-accessible from EL2 and EL1 Secure.

RES0, [6]

RES0 Reserved.

ERXPFGEN, [5]

Error Record Registers enable. The possible values are:

0 ERXPFG* are not write-accessible from EL2 and EL1 Secure. This is the reset value.

1 ERXPFG* are write-accessible from EL2 and EL1 Secure.

AMEN, [4]

Activity Monitor enable. The possible values are:

0 Accesses from EL2, EL1 and EL0 to activity monitor registers are trapped to EL3.

1 Accesses from EL2, EL1 and EL0 to activity monitor registers are not trapped to EL2.

RES0, [3:2]

RES0 Reserved.

ECTLREN, [1]

Extended Control Registers enable. The possible values are:

0 CPUECTLR and CLUSTERECTLR are not write-accessible from EL2 and EL1 Secure. This is the reset value.

1 CPUECTLR and CLUSTERECTLR are write-accessible from EL2 and EL1 Secure.

ACTLREN, [0]

Auxiliary Control Registers enable. The possible values are:

0 CPUACTLR, CPUACTLR2, and CLUSTERACTLR are not write-accessible from EL2 and EL1 Secure. This is the reset value.

1 CPUACTLR, CPUACTLR2, and CLUSTERACTLR are write-accessible from EL2 and EL1 Secure.

Configurations

AArch64 System register ACTLR_EL3 bits [31:0] is mapped to AArch32 register ACTLR (S). See *B1.5 ACTLR, Auxiliary Control Register* on page B1-136.

AArch64 System register ACTLR_EL3 bits [63:32] is mapped to AArch32 register ACTLR2 (S). See *B1.6 ACTLR2, Auxiliary Control Register 2* on page B1-138.

Bit fields and details that are not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

B2.8 AFSR0_EL1, Auxiliary Fault Status Register 0, EL1

AFSR0_EL1 provides additional IMPLEMENTATION DEFINED fault status information for exceptions that are taken to EL1. In the Cortex-A75 core, no additional information is provided for these exceptions. Therefore this register is not used.

Bit field descriptions

AFSR0_EL1 is a 32-bit register, and is part of:

- The Exception and fault handling registers functional group.
- The IMPLEMENTATION DEFINED functional group.

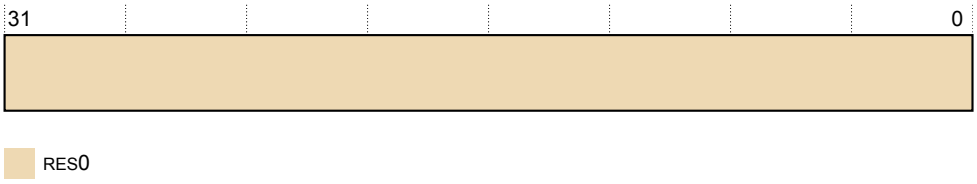

Figure B2-4 AFSR0_EL1 bit assignments

RES0, [31:0]

Reserved, RES0.

Configurations

AArch64 System register AFSR0_EL1 is architecturally mapped to AArch32 System register ADFSR. See *B1.7 ADFSR, Auxiliary Data Fault Status Register* on page B1-139.

Bit fields and details that are not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

B2.9 AFSR0_EL2, Auxiliary Fault Status Register 0, EL2

AFSR0_EL2 provides additional IMPLEMENTATION DEFINED fault status information for exceptions that are taken to EL2. In the A75 core, no additional information is provided for these exceptions. Therefore this register is not used.

Bit field descriptions

AFSR0_EL2 is a 32-bit register, and is part of:

- The Virtualization registers functional group.
- The Exception and fault handling registers functional group.
- The IMPLEMENTATION DEFINED functional group.

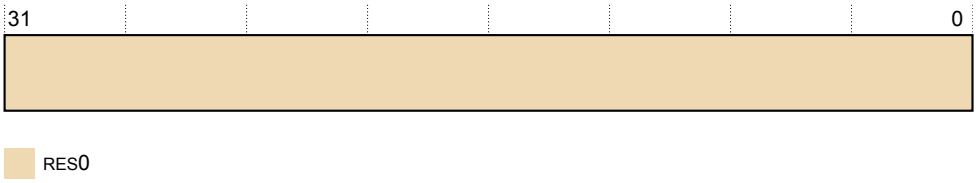

Figure B2-5 AFSR0_EL2 bit assignments

RES0, [31:0]

Reserved, RES0.

Configurations

AArch64 System register AFSR0_EL2 is architecturally mapped to AArch32 System register HADFSR. See *B1.48 HADFSR, Hyp Auxiliary Data Fault Status Syndrome Register* on page B1-199.

Bit fields and details that are not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

B2.10 AFSR0_EL3, Auxiliary Fault Status Register 0, EL3

AFSR0_EL3 provides additional IMPLEMENTATION DEFINED fault status information for exceptions that are taken to EL3. In the Cortex-A75 core, no additional information is provided for these exceptions. Therefore this register is not used.

Bit field descriptions

AFSR0_EL3 is a 32-bit register, and is part of:

- The Exception and fault handling registers functional group.
- The Security registers functional group.
- The IMPLEMENTATION DEFINED functional group.

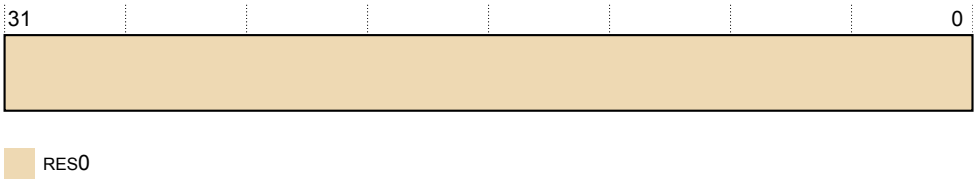

Figure B2-6 AFSR0_EL3 bit assignments

RES0, [31:0]

Reserved, RES0.

Configurations

There are no configurations.

Bit fields and details that are not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

B2.11 AFSR1_EL1, Auxiliary Fault Status Register 1, EL1

AFSR1_EL1 provides additional IMPLEMENTATION DEFINED fault status information for exceptions that are taken to EL1. This register is not used in Cortex-A75.

Bit field descriptions

AFSR1_EL1 is a 32-bit register, and is part of:

- The Exception and fault handling registers functional group.
- The IMPLEMENTATION DEFINED functional group.

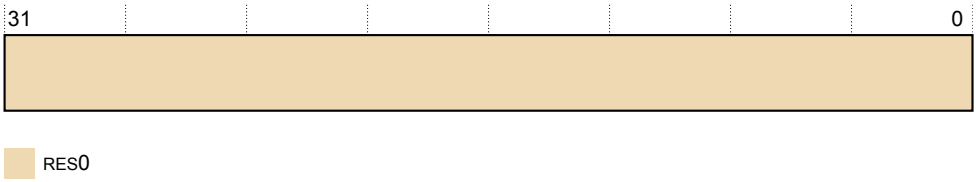

Figure B2-7 AFSR1_EL1 bit assignments

RES0, [31:0]

Reserved, RES0.

Configurations

AFSR1_EL1 is architecturally mapped to AArch32 register AIFSR. See *B1.9 AIFSR, Auxiliary Instruction Fault Status Register* on page B1-141.

Bit fields and details that are not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

B2.12 AFSR1_EL2, Auxiliary Fault Status Register 1, EL2

AFSR1_EL2 provides additional IMPLEMENTATION DEFINED fault status information for exceptions that are taken to EL2. This register is not used in the Cortex-A75 core.

Bit field descriptions

AFSR1_EL2 is a 32-bit register, and is part of:

- The Virtualization registers functional group.
- The Exception and fault handling registers functional group.
- The IMPLEMENTATION DEFINED functional group.

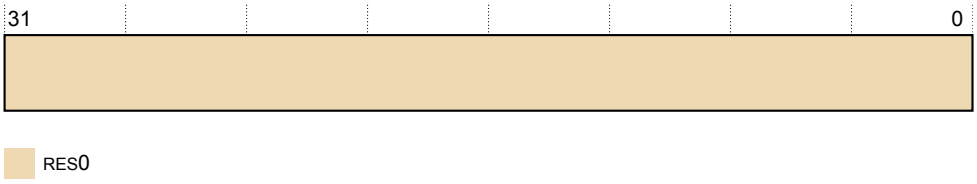

Figure B2-8 AFSR1_EL2 bit assignments

RES0, [31:0]

Reserved, RES0.

Configurations

AArch64 System register AFSR1_EL2 is architecturally mapped to AArch32 System register HAIFSR. See *B1.49 HAIFSR, Hyp Auxiliary Instruction Fault Status Syndrome Register* on page B1-200.

Bit fields and details that are not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

B2.13 AFSR1_EL3, Auxiliary Fault Status Register 1, EL3

AFSR1_EL3 provides additional IMPLEMENTATION DEFINED fault status information for exceptions that are taken to EL3. This register is not used in the Cortex-A75 core.

Bit field descriptions

AFSR1_EL3 is a 32-bit register, and is part of:

- The Exception and fault handling registers functional group.
- The Security registers functional group.
- The IMPLEMENTATION DEFINED functional group.

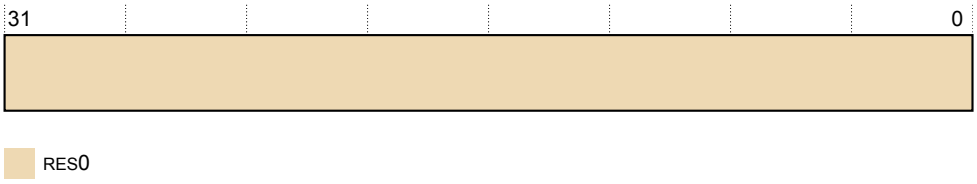

Figure B2-9 AFSR1_EL3 bit assignments

RES0, [31:0]

Reserved, RES0.

Configurations

There are no configuration notes.

Bit fields and details that are not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

B2.14 AIDR_EL1, Auxiliary ID Register, EL1

AIDR_EL1 provides IMPLEMENTATION DEFINED identification information. This register is not used in the A75 core.

Bit field descriptions

AIDR_EL1 is a 32-bit register, and is part of:

- The Identification registers functional group.
- The IMPLEMENTATION DEFINED functional group.

This register is Read Only.

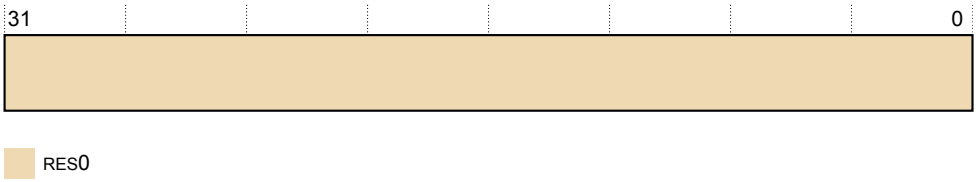

Figure B2-10 AIDR_EL1 bit assignments

RES0, [31:0]

Reserved, RES0.

Configurations

AIDR_EL1 is architecturally mapped to AArch32 register AIDR. See *B1.8 AIDR, Auxiliary ID Register* on page B1-140.

Bit fields and details that are not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

B2.15 AMAIR_EL1, Auxiliary Memory Attribute Indirection Register, EL1

AMAIR_EL1 provides IMPLEMENTATION DEFINED memory attributes for the memory regions specified by MAIR_EL1. This register is not used in the Cortex-A75 core.

Bit field descriptions

AMAIR_EL1 is a 64-bit register, and is part of:

- The Virtual memory control registers functional group.
- The IMPLEMENTATION DEFINED functional group.

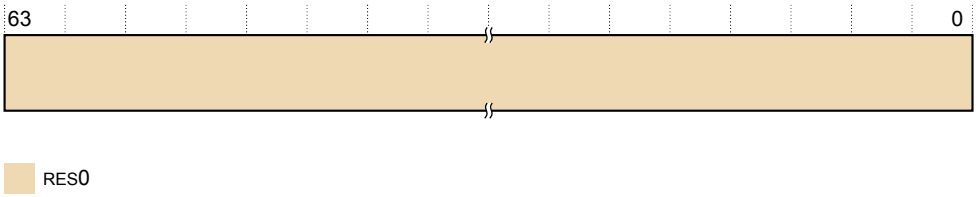

Figure B2-11 AMAIR_EL1 bit assignments

RES0, [63:0]

Reserved, RES0.

Configurations

AArch64 System register AMAIR_EL1 bits [31:0] are architecturally mapped to AArch32 System register AMAIR0. See *B1.10 AMAIR0, Auxiliary Memory Attribute Indirection Register 0* on page B1-142.

AArch64 System register AMAIR_EL1 bits [63:32] are architecturally mapped to AArch32 System register AMAIR1. See *B1.11 AMAIR1, Auxiliary Memory Attribute Indirection Register 1* on page B1-143.

Bit fields and details that are not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

B2.16 AMAIR_EL2, Auxiliary Memory Attribute Indirection Register, EL2

AMAIR_EL2 provides IMPLEMENTATION DEFINED memory attributes for the memory regions specified by MAIR_EL2. This register is not used in the Cortex-A75 core.

Bit field descriptions

AMAIR_EL2 is a 64-bit register, and is part of:

- The Virtualization registers functional group.
- The Virtual memory control registers functional group.
- The IMPLEMENTATION DEFINED functional group.

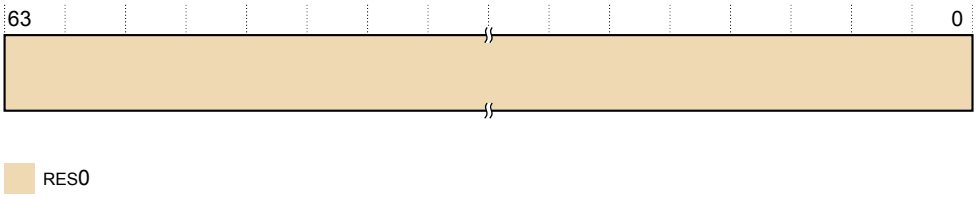

Figure B2-12 AMAIR_EL1 bit assignments

RES0, [63:0]

Reserved, RES0.

Configurations

AArch64 System register AMAIR_EL2 bits [31:0] are architecturally mapped to AArch32 System register HAMAIR0. See *B1.50 HAMAIR0, Hyp Auxiliary Memory Attribute Indirection Register 0* on page B1-201.

AArch64 System register AMAIR_EL2 bits [63:32] are architecturally mapped to AArch32 System register HAMAIR1. See *B1.51 HAMAIR1, Hyp Auxiliary Memory Attribute Indirection Register 1* on page B1-202.

Bit fields and details that are not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

B2.17 AMAIR_EL3, Auxiliary Memory Attribute Indirection Register, EL3

AMAIR_EL3 provides IMPLEMENTATION DEFINED memory attributes for the memory regions specified by MAIR_EL3. This register is not used in the Cortex-A75 core.

Bit field descriptions

AMAIR_EL3 is a 64-bit register, and is part of:

- The Virtual memory control registers functional group.
- The Security registers functional group.
- The IMPLEMENTATION DEFINED functional group.

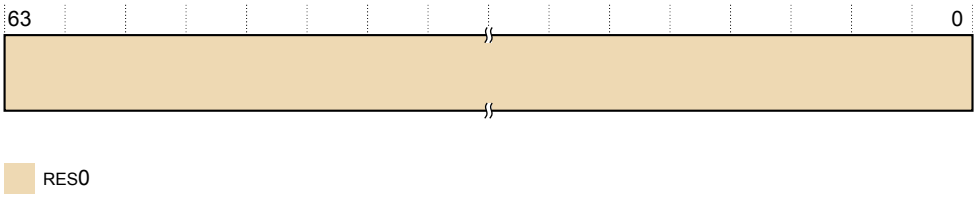

Figure B2-13 AMAIR_EL3 bit assignments

RES0, [63:0]

Reserved, RES0.

Configurations

There are no configurations.

Bit fields and details that are not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

B2.18 CCSIDR_EL1, Cache Size ID Register, EL1

The CCSIDR_EL1 provides information about the architecture of the currently selected cache.

Bit field descriptions

CCSIDR_EL1 is a 32-bit register, and is part of the Identification registers functional group.

This register is Read Only.

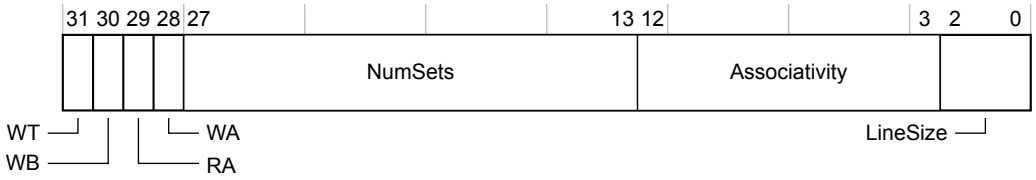

Figure B2-14 CCSIDR_EL1 bit assignments

WT, [31]
Indicates whether the selected cache level supports Write-Through:

`0` Cache Write-Through is not supported at any level.

For more information about encoding, see *CCSIDR_EL1 encodings* on page B2-309.

WB, [30]
Indicates whether the selected cache level supports Write-Back. Permitted values are:

`0` Write-Back is not supported.

`1` Write-Back is supported.

For more information about encoding, see *CCSIDR_EL1 encodings* on page B2-309.

RA, [29]
Indicates whether the selected cache level supports read-allocation. Permitted values are:

`0` Read-allocation is not supported.

`1` Read-allocation is supported.

For more information about encoding, see *CCSIDR_EL1 encodings* on page B2-309.

WA, [28]
Indicates whether the selected cache level supports write-allocation. Permitted values are:

`0` Write-allocation is not supported.

`1` Write-allocation is supported.

For more information about encoding, see *CCSIDR_EL1 encodings* on page B2-309.

NumSets, [27:13]
(Number of sets in cache) - 1. Therefore, a value of 0 indicates one set in the cache. The number of sets does not have to be a power of 2.

For more information about encoding, see *CCSIDR_EL1 encodings* on page B2-309.

Associativity, [12:3]
(Associativity of cache) - 1. Therefore, a value of 0 indicates an associativity of 1. The associativity does not have to be a power of 2.

For more information about encoding, see *CCSIDR_EL1 encodings* on page B2-309.

LineSize, [2:0]
(Log_2(Number of bytes in cache line)) - 4. For example:

Indicates the ($\log_2$ (number of words in cache line)) - 2:

For a line length of 16 bytes: $\text{Log}_2(16) = 4$, LineSize entry = 0. This is the minimum line length.

For a line length of 32 bytes: $\text{Log}_2(32) = 5$, LineSize entry = 1.

For more information about encoding, see *CCSIDR_EL1 encodings* on page B2-309.

Configurations

CCSIDR_EL1 is architecturally mapped to AArch32 register CCSIDR. See *B1.12 CCSIDR, Cache Size ID Register* on page B1-144.

Bit fields and details that are not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

CCSIDR_EL1 encodings

The following table shows the individual bit field and complete register encodings for the CCSIDR_EL1.

Table B2-6 CCSIDR encodings

CSSELR		Cache	Size	Complete register encoding	Register bit field encoding						
Level	InD				WT	WB	RA	WA	NumSets	Associativity	LineSize
0b000	0b0	L1 Data cache	64KB	7007E07A	0	1	1	1	0x003F	0x00F	2
0b000	0b1	L1 Instruction cache	64KB	201FE01A	0	0	1	0	0x00FF	0x003	2
0b001	0b0	L2 cache	256KB	703FE03A	0	1	1	1	0x01FF	0x007	2
			512KB	707FE03A	0	1	1	1	0x03FF	0x007	2
0b001	0b1	Reserved	-	-	-	-	-	-	-	-	-
0b010	0b0	L3 cache	512KB	703FE07A	0	1	1	1	0x01FF	0x00F	2
		When L3 is not implemented, the value of CCSIDR is UNKNOWN.	1024KB	707FE07A					0x03FF		
			2048KB	70FFE07A					0x07FF		
			4096KB	71FFE07A					0x0FFF		
0b010	0b1	Reserved	-	-	-	-	-	-	-	-	-
0b0101 - 0b1111		Reserved	-	-	-	-	-	-	-	-	-

B2.19 CLIDR_EL1, Cache Level ID Register, EL1

The CLIDR_EL1 identifies the type of cache, or caches, implemented at each level, up to a maximum of seven levels.

It also identifies the *Level of Coherency* (LoC) and *Level of Unification* (LoU) for the cache hierarchy.

Bit field descriptions

CLIDR_EL1 is a 64-bit register, and is part of the Identification registers functional group.

This register is Read Only.

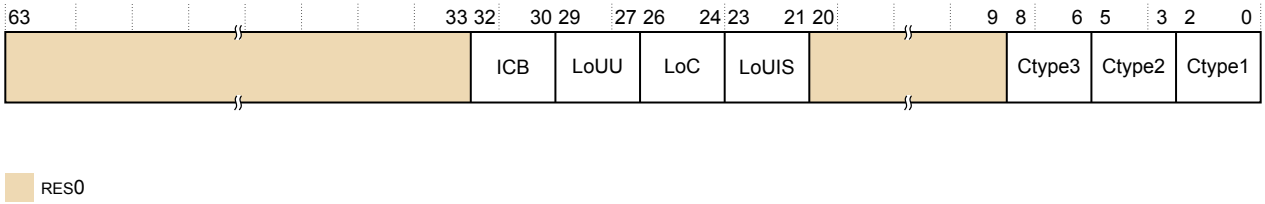

Figure B2-15 CLIDR_EL1 bit assignments

RES0, [63:33]

RES0 Reserved.

ICB, [32:30]

Inner cache boundary. This field indicates the boundary between the inner and the outer domain:

0b10 L2 cache is the highest inner level.

0b11 L3 cache is the highest inner level.

LoUU, [29:27]

Indicates the Level of Unification Uniprocessor for the cache hierarchy:

0b000 No levels of cache need to cleaned or invalidated when cleaning or invalidating to the Point of Unification. This is the value if no cache are configured.

LoC, [26:24]

Indicates the Level of Coherency for the cache hierarchy:

0b010 L3 cache is not implemented.

0b011 L3 cache is implemented.

LoUIS, [23:21]

Indicates the *Level of Unification Inner Shareable* (LoUIS) for the cache hierarchy.

0b000 No cache level needs cleaning to Point of Unification.

RES0, [20:9]

No cache at levels L7 down to L4.

RES0 Reserved.

Ctype3, [8:6]

Indicates the type of cache if the core implements L3 cache. If present, unified instruction and data caches at Level 3:

0b000 Both per-core L2 and cluster L3 caches are present.

0b100 All other options.

Ctype2, [5:3]

Indicates the type of unified instruction and data caches at level 2:

`0b100` L2 cache is implemented as a unified cache.

Ctype1, [2:0]

Indicates the type of cache implemented at L1:

`0b011` Separate instruction and data caches at L1.

Configurations

CLIDR_EL1 is architecturally mapped to AArch32 register CLIDR. See *B1.13 CLIDR, Cache Level ID Register* on page B1-146.

Bit fields and details that are not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

B2.20 CPACR_EL1, Architectural Feature Access Control Register, EL1

The CPACR_EL1 controls access to trace functionality and access to registers associated with Advanced SIMD and floating-point execution.

Bit field descriptions

CPACR_EL1 is a 32-bit register, and is part of the Other system control registers functional group.

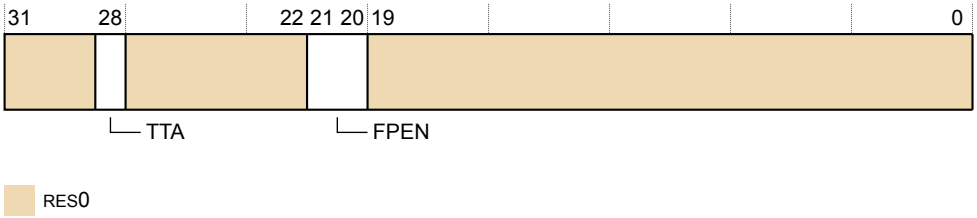

Figure B2-16 CPACR_EL1 bit assignments

RES0, [31:29]

RES0 Reserved.

TTA, [28]

Traps EL0 and EL1 System register accesses to all implemented trace registers to EL1, from both Execution states. This bit is RES0. The core does not provide System Register access to ETM control.

Configurations

CPACR_EL1 is architecturally mapped to AArch32 register CPACR. See *B1.14 CPACR, Architectural Feature Access Control Register* on page B1-148.

Bit fields and details that are not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

B2.21 CPTR_EL2, Architectural Feature Trap Register, EL2

The CPTR_EL2 controls trapping to EL2 for accesses to CPACR, Trace functionality and registers associated with Advanced SIMD and floating-point execution. It also controls EL2 access to this functionality.

Bit field descriptions

CPTR_EL2 is a 32-bit register, and is part of the Virtualization registers functional group.

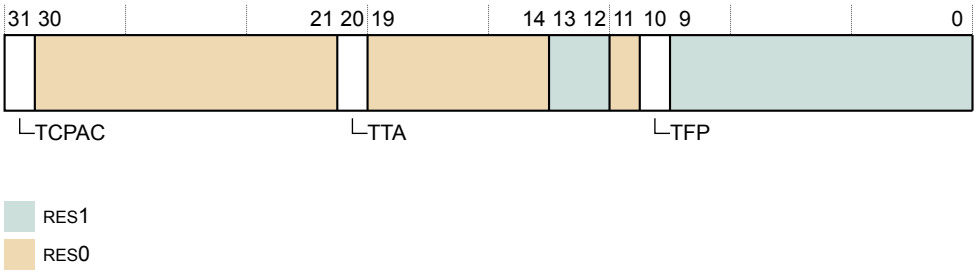

Figure B2-17 CPTR_EL2 bit assignments

TTA, [20]

Trap Trace Access.

This bit is not implemented. RES0.

Configurations

CPTR_EL2 is architecturally mapped to AArch32 register HCPTR.

RW fields in this register reset to UNKNOWN values.

Bit fields and details that are not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

B2.22 CPTR_EL3, Architectural Feature Trap Register, EL3

The CPTR_EL3 controls trapping to EL3 of access to CPACR_EL1, CPTR_EL2, trace functionality and registers associated with Advanced SIMD and floating-point execution.

It also controls EL3 access to trace functionality and registers associated with Advanced SIMD and floating-point execution.

Bit field descriptions

CPTR_EL3 is a 32-bit register, and is part of the Security registers functional group.

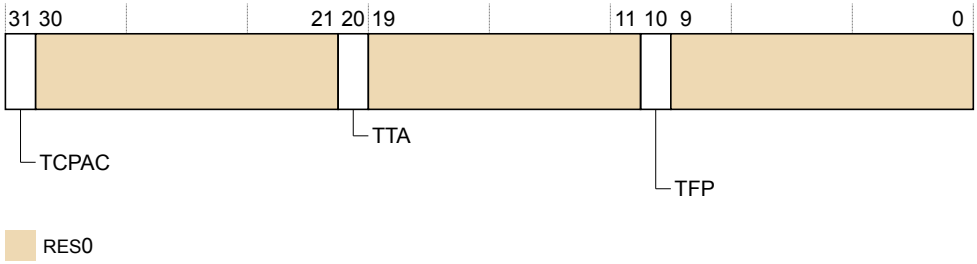

Figure B2-18 CPTR_EL3 bit assignments

TTA, [20]

Trap Trace Access.

Not implemented. RES0.

TFP, [10]

Traps all accesses to SVE, Advanced SIMD and floating-point functionality to EL3. This applies to all Exception levels, both Security states, and both Execution states. The possible values are:

`0` Does not cause any instruction to be trapped. This is the reset value.

`1` Any attempt at any Exception level to execute an instruction that uses the registers that are associated with SVE, Advanced SIMD and floating-point is trapped to EL3, subject to the exception prioritization rules.

Configurations

There are no configuration notes.

Bit fields and details that are not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

B2.23 CPUACTLR_EL1, CPU Auxiliary Control Register, EL1

The CPUACTLR_EL1 provides IMPLEMENTATION DEFINED configuration and control options for the core.

Bit field descriptions

CPUACTLR_EL1 is a 64-bit register, and is part of the IMPLEMENTATION DEFINED registers functional group.

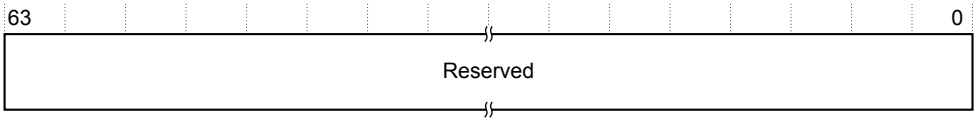

Figure B2-19 CPUACTLR_EL1 bit assignments

Reserved, [63:0]

Reserved for Arm internal use.

Configurations

CPUACTLR_EL1 is:

- Common to the Secure and Non-secure states.
- Mapped to the AArch32 CPUACTLR register. See *B1.15 CPUACTLR, CPU Auxiliary Control Register* on page B1-149.

Usage constraints

Accessing the CPUACTLR_EL1

The CPU Auxiliary Control Register can be written only when the system is idle. Arm recommends that you write to this register after a powerup reset, before the MMU is enabled.

Setting many of these bits can cause significantly lower performance on your code. Therefore, Arm strongly recommends that you do not modify this register unless directed by Arm.

This register is accessible as follows:

This register can be read with the MRS instruction using the following syntax:

```
MRS <Xt>,<systemreg>
```

This register can be written with the MSR instruction using the following syntax:

```
MSR <systemreg>, <Xt>
```

This syntax is encoded with the following settings in the instruction encoding:

<systemreg>	op0	op1	CRn	CRm	op2
S3_0_C15_C1_0	11	000	1111	0001	000

Accessibility

This register is accessible in software as follows:

<systemreg>	Control			Accessibility			
	E2H	TGE	NS	EL0	EL1	EL2	EL3
S3_0_C15_C1_0	x	x	0	-	RW	n/a	RW
S3_0_C15_C1_0	x	0	1	-	RW	RW	RW
S3_0_C15_C1_0	x	1	1	-	n/a	RW	RW

This register is write-accessible in EL1 on either of these conditions:

- ACTLR_EL3.CPUACTLR_EN ==1 && ACTLR_EL2.CPUACTLR_EN==1.
- ACTLR_EL3.CPUACTLR_EN==1 && SCR.NS==0.

This register is write-accessible in EL2 if ACTLR_EL3.CPUACTLR_EN==1.

If write access is not possible, then trap to the lowest Exception level that denied the access (EL2 or EL3).

'n/a' Not accessible. The core cannot be executing at this Exception level, so this access is not possible.

Traps and enables

For a description of the prioritization of any generated exceptions, see *Synchronous exception prioritization* in the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

B2.24 CPUACTLR2_EL1, CPU Auxiliary Control Register 2, EL1

The CPUACTLR2_EL1 provides IMPLEMENTATION DEFINED configuration and control options for the core.

Bit field descriptions

CPUACTLR2_EL1 is a 64-bit register, and is part of the IMPLEMENTATION DEFINED registers functional group.

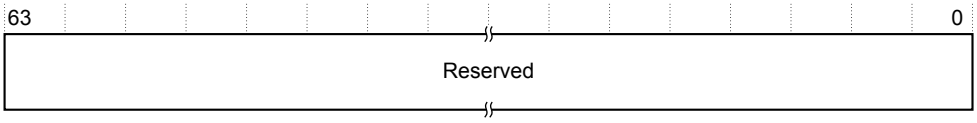

Figure B2-20 CPUACTLR2_EL1 bit assignments

Reserved, [63:0]
Reserved for Arm internal use.

Configurations
CPUACTLR2_EL1 is:

- Common to the Secure and Non-secure states.
- Mapped to the AArch32 CPUACTLR2 register. See *B1.16 CPUACTLR2, CPU Auxiliary Control Register 2* on page B1-151.

Usage constraints

Accessing the CPUACTLR2_EL1
The CPUACTLR2_EL1 can be written only when the system is idle. Arm recommends that you write to this register after a powerup reset, before the MMU is enabled.

Setting many of these bits can cause significantly lower performance on your code. Therefore, Arm strongly recommends that you do not modify this register unless directed by Arm.

This register can be read using MRS with the following syntax:

```
MRS <Xt>,<systemreg>
```

This register can be written using MSR with the following syntax:

```
MSR <systemreg>, <Xt>
```

This syntax is encoded with the following settings in the instruction encoding:

<systemreg>	Op0	CRn	Op1	CRm	Op2
S3_0_C15_C1_1	3	c15	0	c1	1

Accessibility
This register is accessible in software as follows:

<syntax>	Control			Accessibility			
	E2H	TGE	NS	EL0	EL1	EL2	EL3
S3_0_C15_C1_1	x	x	0	-	RW	n/a	RW
S3_0_C15_C1_1	x	0	1	-	RW	RW	RW
S3_0_C15_C1_1	x	1	1	-	n/a	RW	RW

'n/a' Not accessible. The PE cannot be executing at this Exception level, so this access is not possible.

Traps and enables

For a description of the prioritization of any generated exceptions, see *Exception priority order* in the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile* for exceptions taken to AArch64 state, and see *Synchronous exception prioritization* for exceptions taken to AArch64 state.

Write access to this register from EL1 or EL2 depends on the value of bit[0] of ACTLR_EL2 and ACTLR_EL3.

B2.25 CPUCFR_EL1, CPU Configuration Register, EL1

The CPUCFR provides configuration information for the core.

Bit field descriptions

CPUCFR_EL1 is a 32-bit register, and is part of the IMPLEMENTATION DEFINED registers functional group.

This register is Read Only.

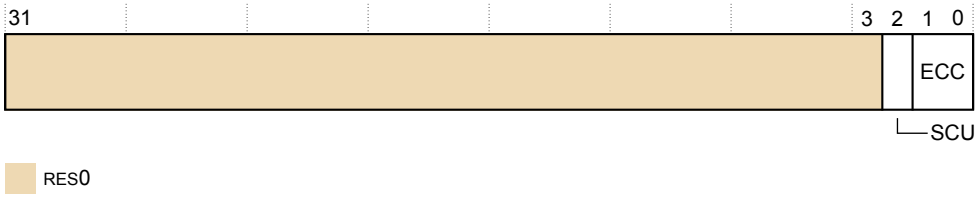

Figure B2-21 CPUCFR_EL1 bit assignments

RES0, [31:3]

Reserved, RES0.

SCU, [2]

Indicates whether the SCU is present or not. The value is:

`0` The SCU is present.

ECC, [1:0]

Indicates whether ECC is present or not. The possible values are:

`00` ECC is not present.

`01` ECC is present.

Configurations

CPUCFR_EL1 is architecturally mapped to AArch32 register CPUCFR. See *B1.17 CPUCFR, CPU Configuration Register* on page B1-153.

Bit fields and details that are not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

Usage constraints

Accessing the CPUCFR_EL1

This register can be read with the MRS instruction using the following syntax:

```
MRS <Xt>,<systemreg>
```

To access the CPUCFR_EL1:

```
MRS <Xt>, CPUCFR_EL1 ; Read CPUCFR_EL1 into Xt
```

This syntax is encoded with the following settings in the instruction encoding:

<systemreg>	op0	op1	CRn	CRm	op2
S3_0_C15_C0_0	11	000	1111	0000	000

Accessibility

This register is accessible in software as follows:

<systemreg>	Control			Accessibility			
	E2H	TGE	NS	EL0	EL1	EL2	EL3
S3_0_C15_C0_0	x	x	0	-	RO	n/a	RO
S3_0_C15_C0_0	x	0	1	-	RO	RO	RO
S3_0_C15_C0_0	x	1	1	-	n/a	RO	RO

'n/a' Not accessible. The PE cannot be executing at this Exception level, so this access is not possible.

B2.26 CPUECTLR_EL1, CPU Extended Control Register, EL1

The CPUECTLR_EL1 provides additional IMPLEMENTATION DEFINED configuration and control options for the core.

Bit field descriptions

CPUECTLR_EL1 is a 64-bit register, and is part of the 64-bit registers functional group.

This register resets to value `0x000000000000FFF0`.

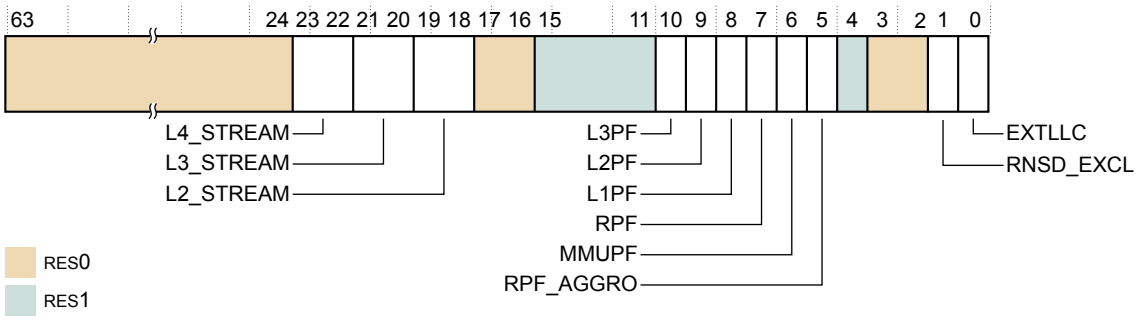

Figure B2-22 CPUECTLR_EL1 bit assignments

RES0, [63:24]

RES0 Reserved.

L4_STREAM, [23:22]

Threshold for direct stream to L4 cache on store. The possible values are:

`0b00` 512KB.

`0b01` 1024KB.

`0b10` 2048KB.

`0b11` Stream disabled.

L3_STREAM, [21:20]

Threshold for direct stream to L3 cache on store. The possible values are:

`0b00` 64KB.

`0b01` 256KB.

`0b10` 512KB.

`0b11` Stream disabled.

L2_STREAM, [19:18]

Threshold for direct stream to L2 cache on store. The possible values are:

`0b00` 16KB.

`0b01` 64KB.

`0b10` 128KB.

`0b11` Stream disabled.

RES0, [17:16]

RES0 Reserved.

RES1, [15:11]

RES1 Reserved.

L3PF, [10]

Enable L3 prefetch requests sent by the stride prefetcher. The possible values are:

0 L3 prefetch requests are disabled.

1 L3 prefetch requests are enabled. This is the reset value.

L2PF, [9]

Enable L2 prefetch requests sent by the stride prefetcher. The possible values are:

0 L2 prefetch requests are disabled.

1 L2 prefetch requests are enabled. This is the reset value.

L1PF, [8]

Enable L1 prefetch requests sent by the stride prefetcher. The possibel values are:

0 L1 prefetch requests are disabled.

1 L1 prefetch requests are enabled. This is the reset value.

RPF, [7]

Enable L2 region prefetch requests. The possible values are:

0 L2 region prefetch requests are disabled.

1 L2 region prefetch requests are enabled. This is the reset value.

MMUPF, [6]

Enable MMU prefetch requests. The possible values are:

0 MMU prefetch requests are disabled.

1 MMU prefetch requests are enabled. This is the reset value.

RPF_AGGRO, [5]

L2 region prefetcher aggressibity. The possible values are:

0 The L2 region prefetcher is less aggressive, with a longer learning phase.

1 The L2 region prefetcher is more aggressive, with a shorter learning phase. This is the reset value.

RES1, [4]

RES1 Reserved.

RES0, [3:2]

RES0 Reserved.

RNSD_EXCL, [1]

Enables signaling of cacheable Exclusive loads on the internal interface between the core and the DSU. The possible values are:

`0x0` Cacheable Exclusive Loads do not use the Exclusive attribute on the internal interface between the core and the DSU. This is the reset value.

`0x1` Cacheable Exclusive Loads use the Exclusive attribute on the internal interface between the core and the DSU.

EXTLLC, [0]

The possible values are:

`0x0` Indicates that an internal Last-level cache is present in the system, and that the DataSource field on the master CHI interface indicates when data is returned from the LLC. This is used to control how the LL_CACHE* PMU events count. This is teh reset value.

`0x1` Indicates that an external Last-level cache is present in the system, and that the DataSource field on the master CHI interface indicates when data is returned from the LLC. This is used to control how the LL_CACHE* PMU events count.

Configurations

The AArch64 CPUECTLR_EL1 register is mapped to the AArch32 CPUECTLR register. See *B1.18 CPUECTLR, CPU Extended Control Register* on page B1-155.

Usage constraints

Accessing the CPUECTLR_EL1

This register can be read using MRS with the following syntax:

```
MRS <Xt>,<systemreg>
```

This register can be written using MSR with the following syntax:

```
MSR <systemreg>, <Xt>
```

This syntax is encoded with the following settings in the instruction encoding:

<systemreg>	op0	op1	CRn	CRm	op2
S3_0_C15_C1_4	11	000	1111	0001	100

Accessibility

This register is accessible in software as follows:

<systemreg>	Control			Accessibility			
	E2H	TGE	NS	EL0	EL1	EL2	EL3
S3_0_C15_C1_4	x	x	0	-	RW	n/a	RW
S3_0_C15_C1_4	x	0	1	-	RW	RW	RW
S3_0_C15_C1_4	x	1	1	-	n/a	RW	RW

'n/a' Not accessible. The PE cannot be executing at this Exception level, so this access is not possible.

Traps and enables

For a description of the prioritization of any generated exceptions, see *Exception priority order* in the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile* for exceptions taken to AArch32 state, and see *Synchronous exception prioritization* for exceptions taken to AArch64 state.

Access to this register depends on bit[1] of ACTLR_EL2 and ACTLR_EL3.

B2.27 CPUPCR_EL3, CPU Private Control Register, EL3

The CPUPCR_EL3 provides IMPLEMENTATION DEFINED configuration and control options for the core.

Bit field descriptions

CPUPCR_EL3 is a 64-bit register, and is part of the Implementation defined registers functional group.

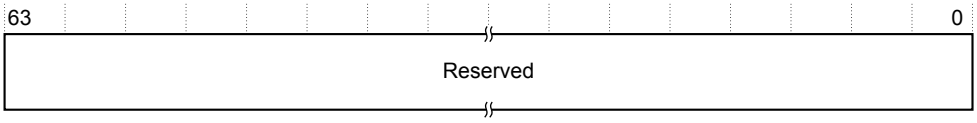

Figure B2-23 CPUPCR_EL3 bit assignments

Reserved, [63:0]

Reserved for Arm internal use.

Configurations

CPUPCR_EL3 is:

- Only accessible in Secure state.
- Mapped to the AArch32 CPUPCR register. See *B1.19 CPUPCR, CPU Private Control Register* on page B1-158.

Usage constraints

Accessing the CPUPCR_EL3

The CPUPCR_EL3 can be written only when the system is idle. Arm recommends that you write to this register after a powerup reset, before the MMU is enabled.

Writing to this register might cause UNPREDICTABLE behaviors. Therefore, Arm strongly recommends that you do not modify this register unless directed by Arm.

This register is accessible as follows:

This register can be read with the MRS instruction using the following syntax:

```
MRS <Xt>,<systemreg>
```

This register can be written with the MSR instruction using the following syntax:

```
MSR <systemreg>, <Xt>
```

This syntax is encoded with the following settings in the instruction encoding:

<systemreg>	op0	op1	CRn	CRm	op2
S3_6_C15_8_1	11	110	1111	1000	001

Accessibility

This register is accessible in software as follows:

<systemreg>	Control			Accessibility			
	E2H	TGE	NS	EL0	EL1	EL2	EL3
S3_6_C15_8_1	x	x	0	-	-	n/a	RW
S3_6_C15_8_1	x	0	1	-	-	-	RW
S3_6_C15_8_1	x	1	1	-	n/a	-	RW

'n/a' Not accessible. The core cannot be executing at this Exception level, so this access is not possible.

Traps and enables

For a description of the prioritization of any generated exceptions, see *Synchronous exception prioritization* in the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

B2.28 CPUPMR_EL3, CPU Private Mask Register, EL3

The CPUPMR_EL3 provides IMPLEMENTATION DEFINED configuration and control options for the core.

Bit field descriptions

CPUPMR_EL3 is a 64-bit register, and is part of the Implementation defined registers functional group.

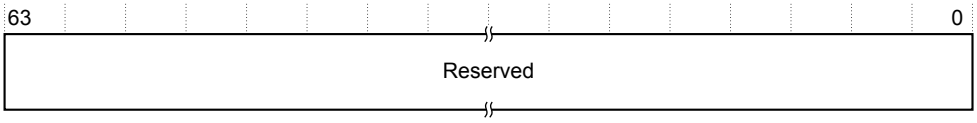

Figure B2-24 CPUPMR_EL3 bit assignments

Reserved, [63:0]

Reserved for Arm internal use.

Configurations

CPUPMR_EL3 is:

- Only accessible from Secure state.
- Mapped to the AArch32 CPUPMR register. See *B1.20 CPUPMR, CPU Private Mask Register* on page B1-160.

Usage constraints

Accessing the CPUPMR_EL3

The CPUPMR_EL3 can be written only when the system is idle. Arm recommends that you write to this register after a powerup reset, before the MMU is enabled.

Writing to this register might cause UNPREDICTABLE behaviors. Therefore, Arm strongly recommends that you do not modify this register unless directed by Arm.

This register is accessible as follows:

This register can be read with the MRS instruction using the following syntax:

```
MRS <Xt>,<systemreg>
```

This register can be written with the MSR instruction using the following syntax:

```
MSR <systemreg>, <Xt>
```

This syntax is encoded with the following settings in the instruction encoding:

<systemreg>	op0	op1	CRn	CRm	op2
S3_6_C15_8_3	11	110	1111	1000	011

Accessibility

This register is accessible in software as follows:

<systemreg>	Control			Accessibility			
	E2H	TGE	NS	EL0	EL1	EL2	EL3
S3_6_C15_8_3	x	x	0	-	-	n/a	RW
S3_6_C15_8_3	x	0	1	-	-	-	RW
S3_6_C15_8_3	x	1	1	-	n/a	-	RW

'n/a' Not accessible. The core cannot be executing at this Exception level, so this access is not possible.

Traps and enables

For a description of the prioritization of any generated exceptions, see *Synchronous exception prioritization* in the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

B2.29 CPUPOR_EL3, CPU Private Operation Register, EL3

The CPUPOR_EL3 provides IMPLEMENTATION DEFINED configuration and control options for the core.

Bit field descriptions

CPUPOR_EL3 is a 64-bit register, and is part of the Implementation defined registers functional group.

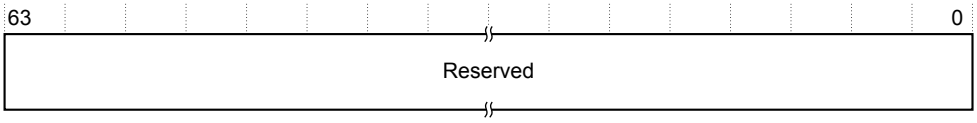

Figure B2-25 CPUPOR_EL3 bit assignments

Reserved, [63:0]

Reserved for Arm internal use.

Configurations

CPUPOR_EL3 is:

- Only accessible in Secure state.
- Mapped to the AArch32 CPUPOR register. See *B1.21 CPUPOR, CPU Private Operation Register* on page B1-162.

Usage constraints

Accessing the CPUPOR_EL3

The CPUPOR_EL3 can be written only when the system is idle. Arm recommends that you write to this register after a powerup reset, before the MMU is enabled.

Writing to this register might cause UNPREDICTABLE behaviors. Therefore, Arm strongly recommends that you do not modify this register unless directed by Arm.

This register is accessible as follows:

This register can be read with the MRS instruction using the following syntax:

```
MRS <Xt>,<systemreg>
```

This register can be written with the MSR instruction using the following syntax:

```
MSR <systemreg>, <Xt>
```

This syntax is encoded with the following settings in the instruction encoding:

<systemreg>	op0	op1	CRn	CRm	op2
S3_6_C15_8_2	11	110	1111	1000	010

Accessibility

This register is accessible in software as follows:

<systemreg>	Control			Accessibility			
	E2H	TGE	NS	EL0	EL1	EL2	EL3
S3_6_C15_8_2	x	x	0	-	-	n/a	RW
S3_6_C15_8_2	x	0	1	-	-	-	RW
S3_6_C15_8_2	x	1	1	-	n/a	-	RW

'n/a' Not accessible. The core cannot be executing at this Exception level, so this access is not possible.

Traps and enables

For a description of the prioritization of any generated exceptions, see *Synchronous exception prioritization* in the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

B2.30 CPUPSELR_EL3, CPU Private Selection Register, EL3

The CPUPSELR_EL3 provides IMPLEMENTATION DEFINED configuration and control options for the core.

Bit field descriptions

CPUPSELR_EL3 is a 32-bit register, and is part of the Implementation defined registers functional group.

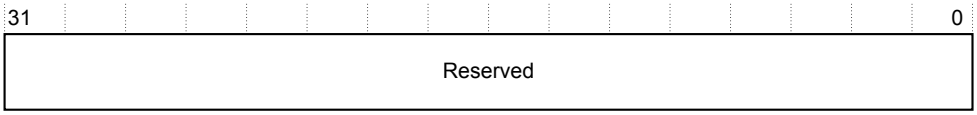

Figure B2-26 CPUPSELR_EL3 bit assignments

Reserved, [31:0]

Reserved for Arm internal use.

Configurations

CPUPSELR_EL3 is:

- Only accessible in Secure state.
- Mapped to the AArch32 CPUPSELR register. See *B1.22 CPUPSELR, CPU Private Selection Register* on page B1-164.

Usage constraints

Accessing the CPUPSELR_EL3

The CPUPSELR_EL3 can be written only when the system is idle. Arm recommends that you write to this register after a powerup reset, before the MMU is enabled.

Writing to this register might cause UNPREDICTABLE behaviors. Therefore, Arm strongly recommends that you do not modify this register unless directed by Arm.

This register is accessible as follows:

This register can be read with the MRS instruction using the following syntax:

```
MRS <Xt>,<systemreg>
```

This register can be written with the MSR instruction using the following syntax:

```
MSR <systemreg>, <Xt>
```

This syntax is encoded with the following settings in the instruction encoding:

<systemreg>	op0	op1	CRn	CRm	op2
S3_6_C15_8_0	11	110	1111	1000	000

Accessibility

This register is accessible in software as follows:

<systemreg>	Control			Accessibility			
	E2H	TGE	NS	EL0	EL1	EL2	EL3
S3_6_C15_8_0	x	x	0	-	-	n/a	RW
S3_6_C15_8_0	x	0	1	-	-	-	RW
S3_6_C15_8_0	x	1	1	-	n/a	-	RW

'n/a' Not accessible. The core cannot be executing at this Exception level, so this access is not possible.

Traps and enables

For a description of the prioritization of any generated exceptions, see *Synchronous exception prioritization* in the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

B2.31 CPUPWRCTLR_EL1, Power Control Register, EL1

The CPUPWRCTLR_EL1 provides information about power control support for the core.

Bit field descriptions

CPUPWRCTLR_EL1 is a 32-bit register, and is part of the IMPLEMENTATION DEFINED registers functional group.

This register resets to value `0x00000000`.

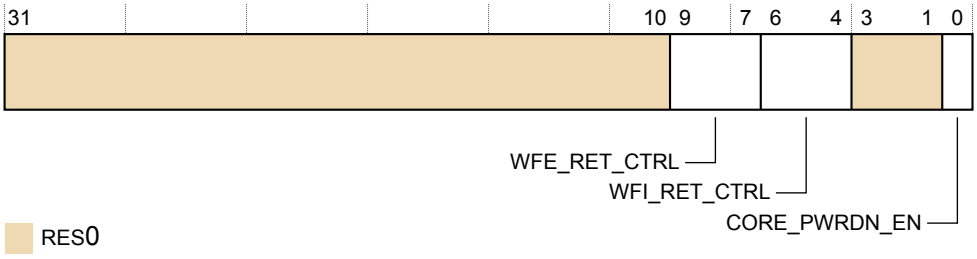

Figure B2-27 CPUPWRCTLR_EL1 bit assignments

RES0, [31:10]

RES0 Reserved.

WFE_RET_CTRL, [9:7]

CPU WFE retention control:

`000` Disable the retention circuit. This is the default value, see *Table B2-7 CPUPWRCTLR Retention Control Field* on page B2-332 for more retention control options.

WFI_RET_CTRL, [6:4]

CPU WFI retention control:

`000` Disable the retention circuit. This is the default value, see *Table B2-7 CPUPWRCTLR Retention Control Field* on page B2-332 for more retention control options.

RES0, [3:1]

RES0 Reserved.

CORE_PWRDN_EN, [0]

Indicates to the power controller using PACTIVE if the core wants to power down when it enters WFI state.

`0` No power down requested.

`0b1` A power down is requested.

Table B2-7 CPUPWRCTLR Retention Control Field

Encoding	System counter ticks required before the core signals retention readiness on PACTIVE to the power controller. The core will not accept a retention entry request until this time.	Minimum retention entry delay (System counter at 50MHz - 10MHz)
`000`	Disable the retention circuit	Default Condition.
`001`	2	40ns – 200ns
`010`	8	160ns – 800ns
`011`	32	640ns – 3,200ns

Table B2-7 CPUPWRCTLR Retention Control Field (continued)

Encoding	System counter ticks required before the core signals retention readiness on PACTIVE to the power controller. The core will not accept a retention entry request until this time.	Minimum retention entry delay (System counter at 50MHz - 10MHz)
100	64	1,280ns – 6,400ns
101	128	2,560ns – 12,800ns
110	256	5,120ns – 25,600ns
111	512	10,240ns – 51,200ns

Configurations

CPUPWRCTLR_EL1 is architecturally mapped to AArch32 register CPUPWRCTLR. See *B1.23 CPUPWRCTLR, CPU Power Control Register* on page B1-166.

Usage constraints

Accessing the CPUPWRCTLR_EL1

This register can be read using MRS with the following syntax:

```
MRS <Xt>,<systemreg>
```

This register can be written using MSR with the following syntax:

```
MSR <systemreg>, <Xt>
```

This syntax is encoded with the following settings in the instruction encoding:

<systemreg>	op0	op1	CRn	CRm	op2
S3_0_C15_C2_7	11	000	1111	0010	111

Accessibility

This register is accessible in software as follows:

<systemreg>	Control			Accessibility			
	E2H	TGE	NS	EL0	EL1	EL2	EL3
S3_0_C15_C2_7	x	x	0	-	RW	n/a	RW
S3_0_C15_C2_7	x	0	1	-	RW	RW	RW
S3_0_C15_C2_7	x	1	1	-	n/a	RW	RW

'n/a' Not accessible. The PE cannot be executing at this Exception level, so this access is not possible.

Traps and enables

For a description of the prioritization of any generated exceptions, see *Exception priority order* in the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile* for exceptions taken to AArch32 state, and see *Synchronous exception prioritization* for exceptions taken to AArch64 state.

Write access to this register from EL1 or EL2 depends on the value of bit[7] of ACTLR_EL2 and ACTLR_EL3.

B2.32 CSSELR_EL1, Cache Size Selection Register, EL1

CSSELR_EL1 selects the current Cache Size ID Register (CCSIDR_EL1), by specifying:

- The required cache level.
- The cache type, either instruction or data cache.

For details of the CCSIDR_EL1, see *B2.18 CCSIDR_EL1, Cache Size ID Register, EL1* on page B2-308.

Bit field descriptions

CSSELR_EL1 is a 32-bit register, and is part of the Identification registers functional group.

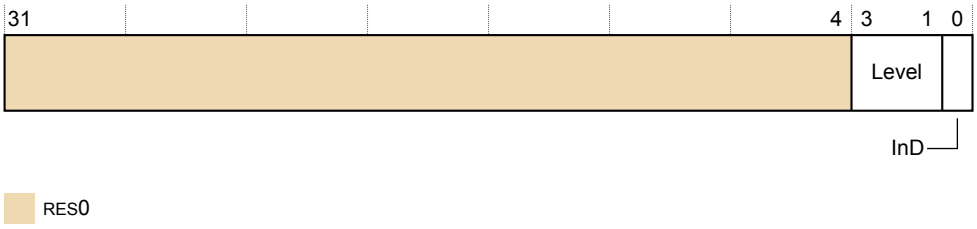

Figure B2-28 CSSELR_EL1 bit assignments

[31:4]

RES0 Reserved.

Level, [3:1]

Cache level of required cache:

`0b000` L1.

`0b001` L2.

`0b010` L3, if present.

The combination of Level=`0b001` and InD=`0b1` is reserved.

The combinations of Level and InD for `0b0100` to `0b1111` are reserved.

InD, [0]

Instruction not Data bit:

`0b0` Data or unified cache.

`0b1` Instruction cache.

The combination of Level=`0b001` and InD=`0b1` is reserved.

The combinations of Level and InD for `0b0100` to `0b1111` are reserved.

Configurations

CSSELR_EL1 is architecturally mapped to AArch32 register CSSELR(NS). See *B1.24 CSSELR, Cache Size Selection Register* on page B1-168.

If a cache level is missing but CSSELR selects the missing cache level, then CCSIDR is RES0.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

B2.33 CTR_EL0, Cache Type Register, EL0

The CTR_EL0 provides information about the architecture of the caches.

Bit field descriptions

CTR_EL0 is a 32-bit register, and is part of the Identification registers functional group.

This register is Read Only.

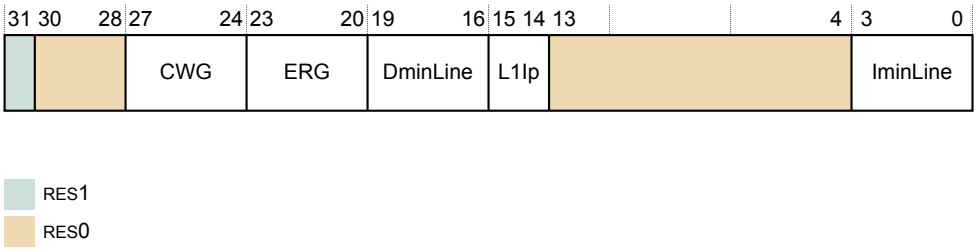

Figure B2-29 CTR_EL0 bit assignments

RES1, [31]

RES1 Reserved.

RES0, [30:28]

RES0 Reserved.

CWG, [27:24]

Cache write-back 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:

`0b0100` Cache write-back granule size is 16 words.

ERG, [23:20]

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:

`0b0100` Exclusive reservation granule size is 16 words.

DminLine, [19:16]

Log_2 of the number of words in the smallest cache line of all the data and unified caches that the core controls:

`0b0100` Smallest data cache line size is 16 words.

VIPT, [15:14]

Instruction cache policy. Indicates the indexing and tagging policy for the L1 Instruction cache:

`0b10` *Virtually Indexed Physically Tagged* (VIPT).

RES0, [13:4]

RES0 Reserved.

IminLine, [3:0]

Log_2 of the number of words in the smallest cache line of all the instruction caches that the core controls.

`0b0100` Smallest instruction cache line size is 16 words.

Configurations

AArch64 System register CTR_EL0 is architecturally mapped to AArch32 register CTR. See *B1.25 CTR, Cache Type Register* on page B1-169.

Bit fields and details that are not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

B2.34 DCZID_EL0, Data Cache Zero ID Register, EL0

The DCZID_EL0 indicates the block size written with byte values of zero by the `DC ZVA` (Data Cache Zero by Address) system instruction.

Bit field descriptions

DCZID_EL0 is a 32-bit register, and is part of the Identification registers functional group.

This register is Read Only.

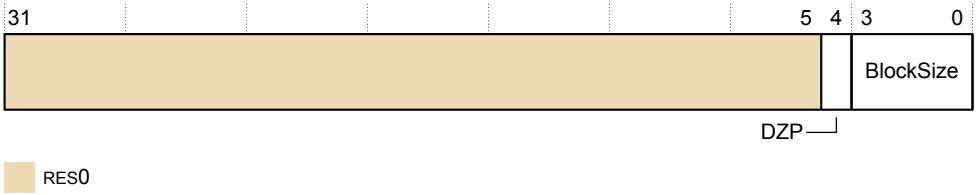

Figure B2-30 DCZID_EL0 bit assignments

RES0, [31:5]

RES0 Reserved.

BlockSize, [3:0]

Log_2 of the block size in words:

`0b0100` The block size is 16 words.

Configurations

There are no configuration notes.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

B2.35 DISR_EL1, Deferred Interrupt Status Register, EL1

The DISR_EL1 records the SError interrupts consumed by an ESB instruction.

Bit field descriptions

DISR_EL1 is a 64-bit register, and is part of the registers *Reliability, Availability, Serviceability* (RAS) functional group.

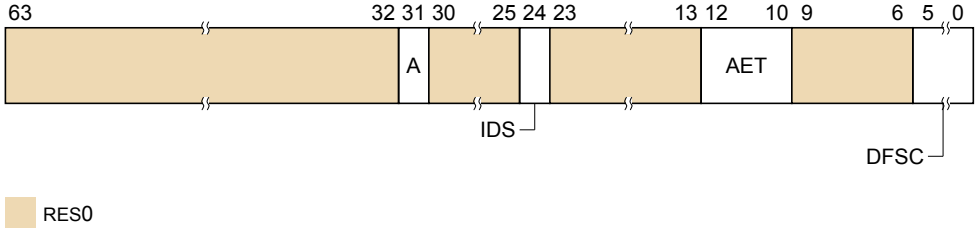

Figure B2-31 DISR_EL1 bit assignments, DISR_EL1.IDS is 0

RES0, [63:32]
Reserved, RES0.

A, [31]
Set to 1 when ESB defers an asynchronous SError interrupt. If the implementation does not include any synchronizable sources of SError interrupt, this bit is RES0.

RES0, [30:25]
Reserved, RES0.

IDS, [24]
Indicates the type of format the deferred SError interrupt uses. The value of this bit is:

0b0 Deferred error uses architecturally-defined format.

RES0, [23:13]
Reserved, RES0.

AET, [12:10]
Asynchronous Error Type. Describes the state of the core after taking an asynchronous Data Abort exception. The only possible value is:

0b000 Uncontainable error (UC).

Note

The recovery software must also examine any implemented fault records to determine the location and extent of the error.

RES0, [9:6]
Reserved, RES0.

DFSC, [5:0]
Data Fault Status Code. This field indicates the type of exception generated. The value is:

0b010001 Asynchronous SError Interrupt.

Configurations
AArch64 System register DISR_EL1 is architecturally mapped to AArch32 register DISR. See *B1.27 DISR, Deferred Interrupt Status Register* on page B1-173.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

B2.36 ERRIDR_EL1, Error ID Register, EL1

The ERRIDR_EL1 defines the number of error record registers.

Bit field descriptions

ERRIDR_EL1 is a 32-bit register, and is part of the registers *Reliability, Availability, Serviceability* (RAS) functional group.

This register is Read Only.

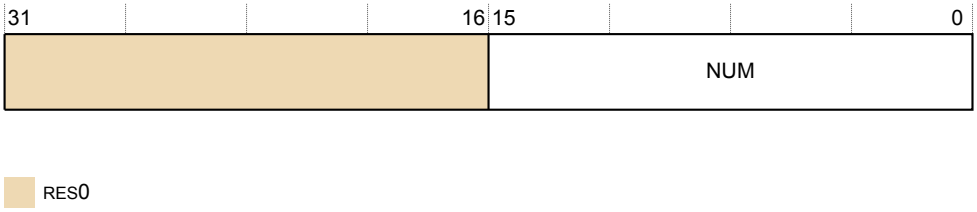

Figure B2-32 ERRIDR_EL1 bit assignments

RES0, [31:16]

RES0 Reserved.

NUM, [15:0]

Number of records that can be accessed through the Error Record system registers.

`0x0002` Two records present.

Configurations

ERRIDR_EL1 is architecturally mapped to AArch32 register ERRIDR. See *B1.28 ERRIDR, Error ID Register* on page B1-176.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

B2.37 ERRSELR_EL1, Error Record Select Register, EL1

The ERRSELR_EL1 selects which error record should be accessed through the Error Record system registers. This register is not reset on a warm reset.

Bit field descriptions

ERRSELR_EL1 is a 64-bit register, and is part of the *Reliability, Availability, Serviceability* (RAS) registers functional group.

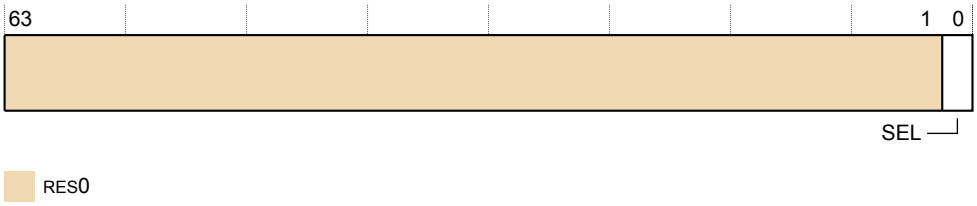

Figure B2-33 ERRSELR_EL1 bit assignments

RES0, [63:1]

Reserved,RES0.

SEL, [0]

Selects which error record should be accessed.

`0` Select record 0 containing errors from Level 1 and Level 2 RAMs located on the Cortex-A75 core.

`1` Select record 1 containing errors from Level 3 RAMs located on the DSU.

Configurations

ERRSELR_EL1 is architecturally mapped to AArch32 register ERRSELR. See *B1.29 ERRSELR, Error Record Select Register* on page B1-177.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

B2.38 ERXADDR_EL1, Selected Error Record Address Register, EL1

Register ERXADDR_EL1 accesses the ERR<n>ADDR address register for the error record selected by ERRSELR_EL1.SEL.

If ERRSELR_EL1.SEL==0, then ERXADDR_EL1 accesses the ERR0ADDR register of the core error record. See *B3.2 ERR0ADDR, Error Record Address Register* on page B3-440.

If ERRSELR_EL1.SEL==1, then ERXADDR_EL1 accesses the ERR1ADDR register of the DSU error record. See the *Arm® DynamIQ™ Shared Unit Technical Reference Manual*.

B2.39 ERXCTLR_EL1, Selected Error Record Control Register, EL1

Register ERXCTLR_EL1 accesses the ERR<n>CTLR control register for the error record selected by ERRSELR_EL1.SEL.

If ERRSELR_EL1.SEL==0, then ERXCTLR_EL1 accesses the ERR0CTLR register of the core error record. See *B3.3 ERR0CTLR, Error Record Control Register* on page B3-441.

If ERRSELR_EL1.SEL==1, then ERXCLTR_EL1 accesses the ERR1CTLR register of the DSU error record. See the *Arm® DynamIQ™ Shared Unit Technical Reference Manual*.

B2.40 ERXFR_EL1, Selected Error Record Feature Register, EL1

Register ERXFR_EL1 accesses the ERR<n>FR feature register for the error record selected by ERRSELR_EL1.SEL.

If ERRSELR_EL1.SEL==0, then ERXFR_EL1 accesses the ERR0FR register of the core error record. See *B3.4 ERR0FR, Error Record Feature Register* on page B3-443.

If ERRSELR_EL1.SEL==1, then ERXFR_EL1 accesses the ERR1FR register of the DSU error record. See the *Arm® DynamIQ™ Shared Unit Technical Reference Manual*.

B2.41 ERXMISC0_EL1, Selected Error Record Miscellaneous Register 0, EL1

Register ERXMISC0_EL1 accesses the ERR<n>MISC0 register for the error record selected by ERRSELR_EL1.SEL.

If ERRSELR_EL1.SEL==0, then ERXMISC0_EL1 accesses the ERR0MISC0 register of the core error record. See *B3.5 ERR0MISC0, Error Record Miscellaneous Register 0* on page B3-445.

If ERRSELR_EL1.SEL==1, then ERXMISC0_EL1 accesses the ERR1MISC0 register of the DSU error record. See the *Arm® DynamIQ™ Shared Unit Technical Reference Manual*.

B2.42 ERXMISC1_EL1, Selected Error Record Miscellaneous Register 1, EL1

Register ERXMISC1_EL1 accesses the ERR<n>MISC1 miscellaneous register 1 for the error record selected by ERRSELR_EL1.SEL.

If ERRSELR_EL1.SEL==0, then ERXMISC1_EL1 accesses the ERR0MISC1 register of the core error record. See *B3.6 ERR0MISC1, Error Record Miscellaneous Register 1* on page B3-447.

If ERRSELR_EL1.SEL==1, then ERXMISC1_EL1 accesses the ERR1MISC1 register of the DSU error record. See the *Arm® DynamIQ™ Shared Unit Technical Reference Manual*.

B2.43 ERXPFGCDNR_EL1, Selected Error Pseudo Fault Generation Count Down Register, EL1

Register ERXPFGCDNR_EL1 accesses the ERR<n>PFGCNDR register for the error record selected by ERRSELR_EL1.SEL.

If ERRSELR_EL1.SEL==0, then ERXPFGCDNR_EL1 accesses the ERR0PFGCDNR register of the core error record. See *B3.7 ERR0PFGCDNR, Error Pseudo Fault Generation Count Down Register* on page B3-448.

If ERRSELR_EL1.SEL==1, then ERXPFGCDNR_EL1 accesses the ERR1PFGCDNR register of the DSU error record. See the *Arm® DynamIQ™ Shared Unit Technical Reference Manual*.

Configurations

ERXPFGCDNR_EL1 is architecturally mapped to AArch32 register ERXPFGCDNR. See *B1.40 ERXPFGCDNR, Selected Error Pseudo Fault Generation Count Down Register* on page B1-188.

Accessing the ERXPFGCDNR_EL1

This register can be read using MRS with the following syntax:

```
MRS <syntax>
```

This register can be written using MSR with the following syntax:

```
MSR <syntax>
```

This syntax is encoded with the following settings in the instruction encoding:

<systemreg>	op0	op1	CRn	CRm	op2
S3_0_C15_C2_2	11	000	1111	0010	010

Accessibility

This register is accessible in software as follows:

<syntax>	Control			Accessibility			
	E2H	TGE	NS	EL0	EL1	EL2	EL3
S3_0_C15_C2_2	x	x	0	-	RW	n/a	RW
S3_0_C15_C2_2	x	0	1	-	RW	RW	RW
S3_0_C15_C2_2	x	1	1	-	n/a	RW	RW

n/a Not accessible. Executing the PE at this exception level is not permitted.

Traps and enables

For a description of the prioritization of any generated exceptions, see *Exception priority order* in the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile* for exceptions taken to AArch32 state, and see *Synchronous exception prioritization* for exceptions taken to AArch64 state. Subject to these prioritization rules, the following traps and enables are applicable when accessing this register.

ERXPFGCDNR_EL1 is accessible at EL3 and can be accessible at EL1 and EL2 depending on the value of bit[5] in ACTLR_EL2 and ACTLR_EL3. See *B2.6 ACTLR_EL2, Auxiliary Control Register, EL2* on page B2-294 and *B2.7 ACTLR_EL3, Auxiliary Control Register, EL3* on page B2-296.

ERXPFGCDNR_EL1 is UNDEFINED at EL0.

If ERXPFGCDNR_EL1 is accessible at EL1 and HCR_EL2.TERR == 1, then direct reads and writes of ERXPFGCDNR_EL1 at Non-secure EL1 generate a Trap exception to EL2.

If ERXPFGCDNR_EL1 is accessible at EL1 or EL2 and SCR_EL3.TERR == 1, then direct reads and writes of ERXPFGCDNR_EL1 at EL1 or EL2 generate a Trap exception to EL3.

B2.44 ERXPFGCTLR_EL1, Selected Error Pseudo Fault Generation Control Register, EL1

Register ERXPFGCTLR_EL1 accesses the ERR<n>PFGCTLR register for the error record selected by ERRSELR_EL1.SEL.

If ERRSELR_EL1.SEL==0, then ERXPFGCTLR_EL1 accesses the ERR0PFGCTLR register of the core error record. See *B3.8 ERR0PFGCTLR, Error Pseudo Fault Generation Control Register* on page B3-449.

If ERRSELR_EL1.SEL==1, then ERXPFGCTLR_EL1 accesses the ERR1PFGCTLR register of the DSU error record. See the *Arm® DynamIQ™ Shared Unit Technical Reference Manual*.

Configurations

ERXPFGCTLR_EL1 is architecturally mapped to AArch32 register ERXPFGCTLR. See *B1.41 ERXPFGCTLR, Selected Error Pseudo Fault Generation Control Register* on page B1-190.

Accessing the ERXPFGCTLR_EL1

This register can be read using MRS with the following syntax:

```
MRS <syntax>
```

This register can be written using MSR with the following syntax:

```
MSR <syntax>
```

This syntax is encoded with the following settings in the instruction encoding:

<systemreg>	op0	op1	CRn	CRm	op2
S3_0_C15_C2_1	11	000	1111	0010	001

Accessibility

This register is accessible in software as follows:

<syntax>	Control			Accessibility			
	E2H	TGE	NS	EL0	EL1	EL2	EL3
S3_0_C15_C2_1	x	x	0	-	RW	n/a	RW
S3_0_C15_C2_1	x	0	1	-	RW	RW	RW
S3_0_C15_C2_1	x	1	1	-	n/a	RW	RW

'n/a' Not accessible. The PE cannot be executing at this Exception level, so this access is not possible.

Traps and enables

For a description of the prioritization of any generated exceptions, see *Exception priority order* in the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile* for exceptions taken to AArch32 state, and see *Synchronous exception prioritization* for exceptions taken to AArch64 state. Subject to these prioritization rules, the following traps and enables are applicable when accessing this register.

ERXPFGCTLR_EL1 is accessible at EL3 and can be accessible at EL1 and EL2 depending on the value of bit[5] in ACTLR_EL2 and ACTLR_EL3. See *B2.6 ACTLR_EL2, Auxiliary Control Register, EL2* on page B2-294 and *B2.7 ACTLR_EL3, Auxiliary Control Register, EL3* on page B2-296.

ERXPFGCTLR_EL1 is UNDEFINED at EL0.

If ERXPFGCTLR_EL1 is accessible at EL1 and HCR_EL2.TERR == 1, then direct reads and writes of ERXPFGCTLR_EL1 at Non-secure EL1 generate a Trap exception to EL2.

If ERXPFGCTLR_EL1 is accessible at EL1 or EL2 and SCR_EL3.TERR == 1, then direct reads and writes of ERXPFGCTLR_EL1 at EL1 or EL2 generate a Trap exception to EL3.

B2.45 ERXPFGFR_EL1, Selected Pseudo Fault Generation Feature Register, EL1

Register ERXPFGFR_EL1 accesses the ERR<n>PFGFR register for the error record selected by ERRSELR_EL1.SEL.

If ERRSELR_EL1.SEL==0, then ERXPFGFR_EL1 accesses the ERR0PFGFR register of the core error record. See *B3.9 ERR0PFGFR, Error Pseudo Fault Generation Feature Register* on page B3-451.

If ERRSELR_EL1.SEL==1, then ERXPFGFR_EL1 accesses the ERR1PFGFR register of the DSU error record. See the *Arm® DynamIQ™ Shared Unit Technical Reference Manual*.

Configurations

ERXPFGFR_EL1 is architecturally mapped to AArch32 register ERXPFGFR. See *B1.42 ERXPFGFR, Selected Pseudo Fault Generation Feature Register* on page B1-192.

Accessing the ERXPFGFR_EL1

This register can be read using MRS with the following syntax:

```
MRS <syntax>
```

This syntax is encoded with the following settings in the instruction encoding:

<systemreg>	op0	op1	CRn	CRm	op2
S3_0_C15_C2_0	11	000	1111	0010	000

Accessibility

This register is accessible in software as follows:

<syntax>	Control			Accessibility			
	E2H	TGE	NS	EL0	EL1	EL2	EL3
S3_0_C15_C2_0	x	x	0	-	RO	n/a	RO
S3_0_C15_C2_0	x	0	1	-	RO	RO	RO
S3_0_C15_C2_0	x	1	1	-	n/a	RO	RO

'n/a' Not accessible. The PE cannot be executing at this Exception level, so this access is not possible.

Traps and enables

For a description of the prioritization of any generated exceptions, see *Exception priority order* in the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile* for exceptions taken to AArch32 state, and see *Synchronous exception prioritization* for exceptions taken to AArch64 state. Subject to these prioritization rules, the following traps and enables are applicable when accessing this register.

ERXPFGR_EL1 is accessible at EL3 and can be accessible at EL1 and EL2 depending on the value of bit[5] in ACTLR_EL2 and ACTLR_EL3. See *B2.6 ACTLR_EL2, Auxiliary Control Register, EL2* on page B2-294 and *B2.7 ACTLR_EL3, Auxiliary Control Register, EL3* on page B2-296.

ERXPFGR_EL1 is UNDEFINED at EL0.

If ERXPFGR_EL1 is accessible at EL1 and HCR_EL2.TERR == 1, then direct reads and writes of ERXPFGR_EL1 at Non-secure EL1 generate a Trap exception to EL2.

If ERXPFGR_EL1 is accessible at EL1 or EL2 and SCR_EL3.TERR == 1, then direct reads and writes of ERXPFGR_EL1 at EL1 or EL2 generate a Trap exception to EL3.

B2.46 ERXSTATUS_EL1, Selected Error Record Primary Status Register, EL1

Register ERXSTATUS_EL1 accesses the ERR<n>STATUS primary status register for the error record selected by ERRSELR_EL1.SEL.

If ERRSELR_EL1.SEL==0, then ERXSTATUS_EL1 accesses the ERR0STATUS register of the core error record. See *B3.10 ERR0STATUS, Error Record Primary Status Register* on page B3-453.

If ERRSELR_EL1.SEL==1, then ERXSTATUS_EL1 accesses the ERR1STATUS register of the DSU error record. See the *Arm® DynamIQ™ Shared Unit Technical Reference Manual*.

B2.47 ESR_EL1, Exception Syndrome Register, EL1

The ESR_EL1 holds syndrome information for an exception taken to EL1.

Bit field descriptions

ESR_EL1 is a 32-bit register, and is part of the Exception and fault handling registers functional group.

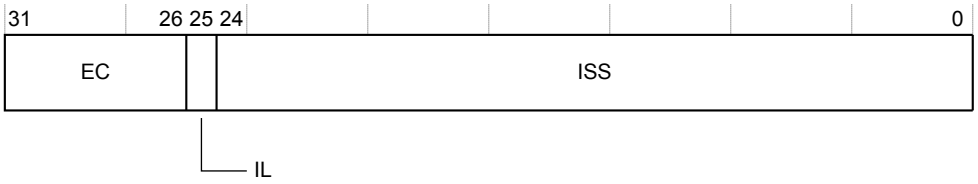

Figure B2-34 ESR_EL1 bit assignments

EC, [31:26]
Exception Class. Indicates the reason for the exception that this register holds information about.

IL, [25]
Instruction Length for synchronous exceptions. The possible values are:

`0` 16-bit.

`1` 32-bit.

This field is `1` for the SError interrupt, instruction aborts, misaligned PC, Stack pointer misalignment, data aborts for which the ISV bit is `0`, exceptions caused by an illegal instruction set state, and exceptions using the `0x00` Exception Class.

ISS, [24:0]
Syndrome information.

When reporting a virtual SEI, bits[24:0] take the value of VSESRL_EL2[24:0].

When reporting a physical SEI, the following occurs:
- IDS==0 (architectural syndrome).
- AET always reports an uncontainable error (UC) with value `0b000` or an unrecoverable error (UEU) with value `0b001`.
- EA is RES0.

When reporting a synchronous data abort, EA is RES0.

See *B2.101 VSESR_EL2, Virtual SError Exception Syndrome Register* on page B2-432.

Configurations
ESR_EL1 is architecturally mapped to AArch32 register DFSR (NS).

B2.48 ESR_EL2, Exception Syndrome Register, EL2

The ESR_EL2 holds syndrome information for an exception taken to EL2.

Bit field descriptions

ESR_EL2 is a 32-bit register, and is part of:

- The Virtualization registers functional group.
- The Exception and fault handling registers functional group.

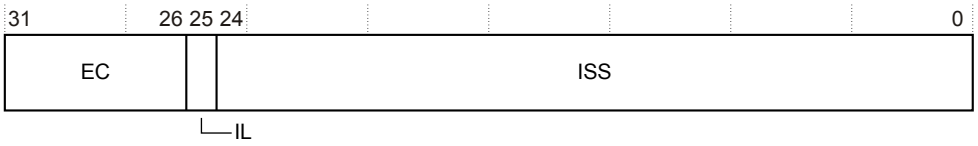

Figure B2-35 ESR_EL2 bit assignments

EC, [31:26]
Exception Class. Indicates the reason for the exception that this register holds information about. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile* for more information.

IL, [25]
Instruction Length for synchronous exceptions. The possible values are:

`0` 16-bit.

`1` 32-bit.

See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile* for more information.

ISS, [24:0]
Syndrome information. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile* for more information.

When reporting a virtual SEI, bits[24:0] take the value of VSESRL_EL2[24:0].

When reporting a physical SEI, the following occurs:
- IDS==0 (architectural syndrome).
- AET always reports an uncontainable error (UC) with value `0b000` or an unrecoverable error (UEU) with value `0b001`.
- EA is RES0.

When reporting a synchronous data abort, EA is RES0.

See *B2.101 VSESR_EL2, Virtual SError Exception Syndrome Register* on page B2-432.

Configurations
ESR_EL2 is architecturally mapped to AArch32 register HSR. See *B1.55 HSR, Hyp Syndrome Register* on page B1-208.

If EL2 is not implemented, this register is RES0 from EL3.

RW fields in this register reset to architecturally UNKNOWN values.

B2.49 ESR_EL3, Exception Syndrome Register, EL3

The ESR_EL3 holds syndrome information for an exception taken to EL3.

Bit field descriptions

ESR_EL3 is a 32-bit register, and is part of the Exception and fault handling registers functional group.

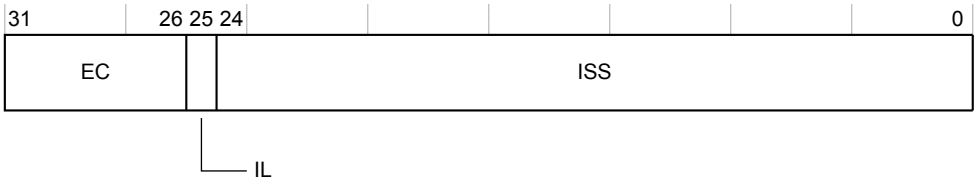

Figure B2-36 ESR_EL3 bit assignments

EC, [31:26]
Exception Class. Indicates the reason for the exception that this register holds information about.

IL, [25]
Instruction Length for synchronous exceptions. The possible values are:

`0` 16-bit.

`1` 32-bit.

This field is `1` for the SError interrupt, instruction aborts, misaligned PC, Stack pointer misalignment, data aborts for which the ISV bit is `0`, exceptions caused by an illegal instruction set state, and exceptions using the `0x0` Exception Class.

ISS, [24:0]
Syndrome information.

When reporting a virtual SEI, bits[24:0] take the value of VSESRL_EL2[24:0].

When reporting a physical SEI, the following occurs:
- IDS==0 (architectural syndrome).
- AET always reports an uncontainable error (UC) with value `0b000` or an unrecoverable error (UEU) with value `0b001`.
- EA is RES0.

When reporting a synchronous data abort, EA is RES0.

See *B2.101 VSESR_EL2, Virtual SError Exception Syndrome Register* on page B2-432.

Configurations
ESR_EL3 is mapped to AArch32 register DFSR(S). See *B1.26 DFSR, Data Fault Status Register* on page B1-171.

RW fields in this register reset to architecturally UNKNOWN values.

B2.50 HACR_EL2, Hyp Auxiliary Configuration Register, EL2

HACR_EL2 controls trapping to EL2 of IMPLEMENTATION DEFINED aspects of Non-secure EL1 or EL0 operation. This register is not used in the Cortex-A75 core.

Bit field descriptions

HACR_EL2 is a 32-bit register, and is part of Virtualization registers functional group.

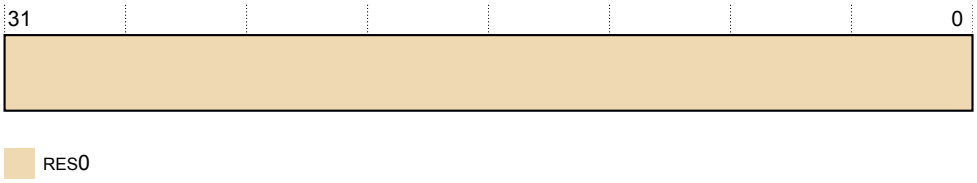

Figure B2-37 HACR_EL2 bit assignments

RES0, [31:0]

Reserved, RES0.

Configurations

AArch64 System register HACR_EL2 is architecturally mapped to AArch32 System register HACR. See *B1.45 HACR, Hyp Auxiliary Configuration Register* on page B1-195.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

B2.51 HCR_EL2, Hypervisor Configuration Register, EL2

The HCR_EL2 provides configuration control for virtualization, including whether various Non-secure operations are trapped to EL2.

Bit field descriptions

HCR_EL2 is a 64-bit register, and is part of the Virtualization registers functional group.

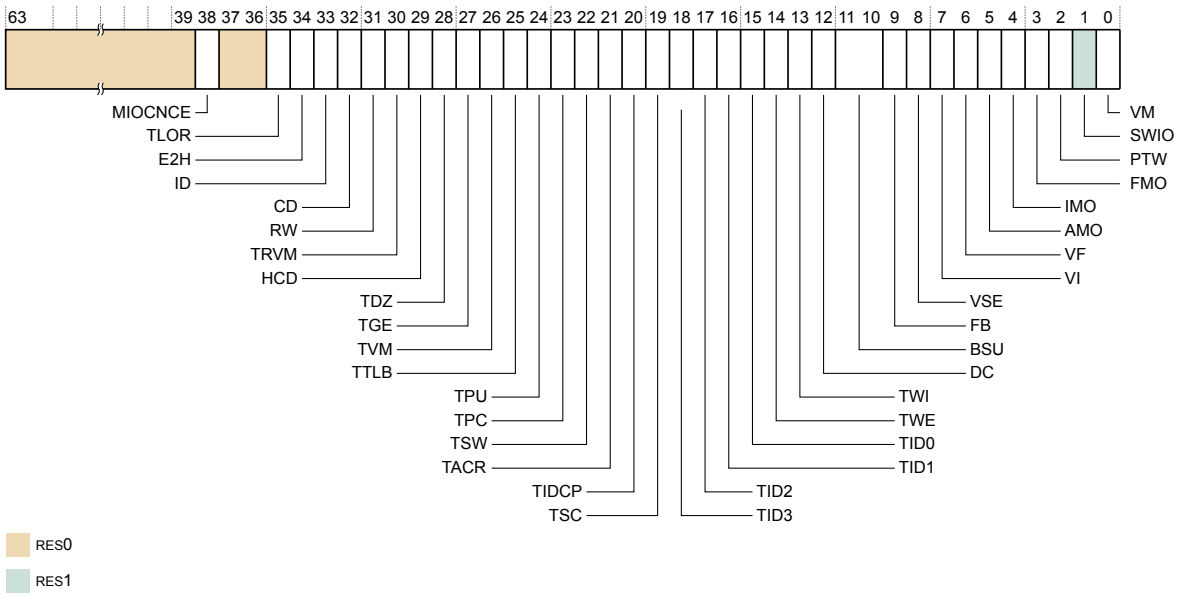

Figure B2-38 HCR_EL2 bit assignments

RES0, [63:39]

RES0 Reserved.

MIOCNCE, [38]

Mismatched Inner/Outer Cacheable Non-Coherency Enable, for the Non-secure EL1 and EL0 translation regime.

This bit is not implemented, RAZ/WI.

RW, [31]

Execution state control for lower Exception levels. The possible values are:

`0` Lower levels are all AArch32.

`1` The Execution state for EL1 is AArch64. The Execution state for EL0 is determined by the current value of PSTATE.nRW when executing at EL0.

HCD, [29]
HVC instruction disable.

This bit is reserved, RES0.

TGE, [27]
Traps general exceptions. If this bit is set, and SCR_EL3.NS is set, then:

- All exceptions that would be routed to EL1 are routed to EL2.
- The SCTLR_EL1.M bit is treated as `0` regardless of its actual state, other than for reading the bit.
- The HCR_EL2.FMO, IMO, and AMO bits are treated as `1` regardless of their actual state, other than for reading the bits.
- All virtual interrupts are disabled.
- Any implementation defined mechanisms for signaling virtual interrupts are disabled.
- An exception return to EL1 is treated as an illegal exception return.

HCR_EL2.TGE must not be cached in a TLB.

When the value of SCR_EL3.NS is 0 the core behaves as if this field is 0 for all purposes other than a direct read or write access of HCR_EL2.

TID3, [18]
Traps ID group 3 registers. The possible values are:

`0` ID group 3 register accesses are not trapped.

`1` Reads to ID group 3 registers executed from Non-secure EL1 are trapped to EL2.

See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile* for the registers covered by this setting.

TID0, [15]
Trap ID Group 0. When `1`, this causes reads to the following registers executed from EL1 or EL0 if not UNDEFINED to be trapped to EL2:

FPSID and JIDR.

When the value of SCR_EL3.NS is `0` the PE behaves as if this field is `0` for all purposes other than a direct read or write access of HCR_EL2.

SWIO, [1]
Set/Way Invalidation Override. Non-secure EL1 execution of the data cache invalidate by set/way instruction is treated as data cache clean and invalidate by set/way.

This bit is RES1.

Configurations
HCR_EL2[31:0] is architecturally mapped to AArch32 register HCR. See *B1.52 HCR, Hyp Configuration Register* on page B1-203.

HCR_EL2[63:32] is architecturally mapped to AArch32 register HCR2. See *B1.53 HCR2, Hyp Configuration Register 2* on page B1-205.

If EL2 is not implemented, this register is RES0 from EL3

RW fields in this register reset to architecturally UNKNOWN values.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

B2.52 ID_AA64AFR0_EL1, AArch64 Auxiliary Feature Register 0

The core does not use this register, ID_AA64AFR0_EL1 is RES0.

B2.53 ID_AA64AFR1_EL1, AArch64 Auxiliary Feature Register 1

The core does not use this register, ID_AA64AFR0_EL1 is RES0.

B2.54 ID_AA64DFR0_EL1, AArch64 Debug Feature Register 0, EL1

Provides top-level information about the debug system in AArch64.

Bit field descriptions

ID_AA64DFR0_EL1 is a 64-bit register, and is part of the Identification registers functional group.

This register is Read Only.

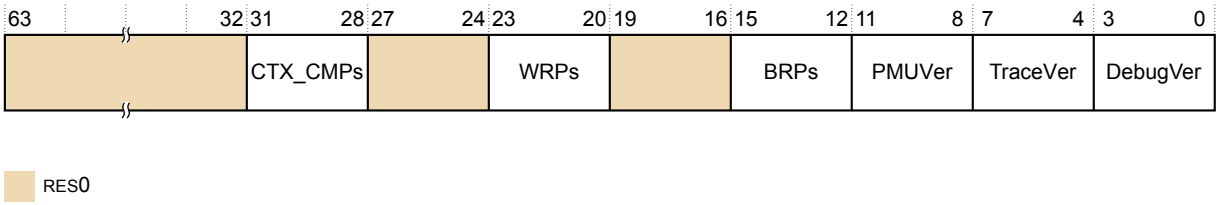

Figure B2-39 ID_AA64DFR0_EL1 bit assignments

RES0, [63:32]

RES0 Reserved.

CTX_CMPs, [31:28]

Number of breakpoints that are context-aware, minus 1. These are the highest numbered breakpoints:

0x1 Two breakpoints are context-aware.

RES0, [27:24]

RES0 Reserved.

WRPs, [23:20]

The number of watchpoints minus 1:

0x3 Four watchpoints.

RES0, [19:16]

RES0 Reserved.

BRPs, [15:12]

The number of breakpoints minus 1:

0x5 Six breakpoints.

PMUVer, [11:8]

Performance Monitors Extension version.

0x4 Performance monitor system registers implemented, PMUv3.

TraceVer, [7:4]

Trace extension:

0x0 Trace system registers not implemented.

DebugVer, [3:0]

Debug architecture version:

0x8 Armv8-A debug architecture implemented.

Configurations

ID_AA64DFR0_EL1 is architecturally mapped to external register EDDFR.

Bit fields and details that are not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

B2.55 ID_AA64DFR1_EL1, AArch64 Debug Feature Register 1, EL1

This register is reserved for future expansion of top level information about the debug system in AArch64 state.

B2.56 ID_AA64ISAR0_EL1, AArch64 Instruction Set Attribute Register 0, EL1

The ID_AA64ISAR0_EL1 provides information about the instructions implemented in AArch64 state, including the instructions that are provided by the Cryptographic Extension.

Bit field descriptions

ID_AA64ISAR0_EL1 is a 64-bit register, and is part of the Identification registers functional group.

This register is Read Only.

The optional Cryptographic Extension is not included in the base product of the core. Arm requires licensees to have contractual rights to obtain the Cryptographic Extension.

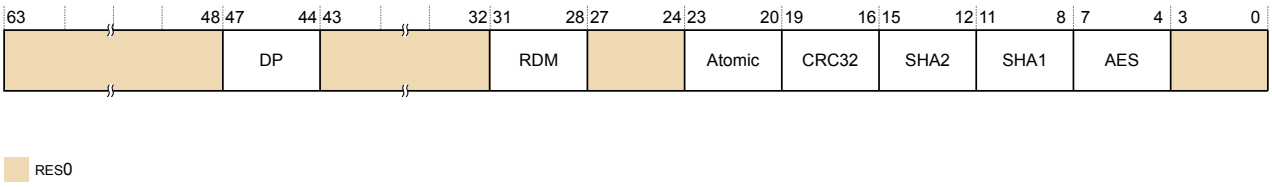

Figure B2-40 ID_AA64ISAR0_EL1 bit assignments

RES0, [63:48]

RES0 Reserved.

DP, [47:44]

Indicates whether Dot Product support instructions are implemented.

0x1 UDOT, SDOT instructions are implemented.

RES0, [43:32]

RES0 Reserved.

RDM, [31:28]

Indicates whether SQRDMLAH and SQRDMLSH instructions in AArch64 are implemented.

0x1 SQRDMLAH and SQRDMLSH instructions implemented.

RES0, [27:24]

RES0 Reserved.

Atomic, [23:20]

Indicates whether Atomic instructions in AArch64 are implemented. The value is:

0x2 LDADD, LDCLR, LDEOR, LDSET, LDSMAX, LDSMIN, LDUMAX, LDUMIN, CAS, CASP, and SWP instructions are implemented.

CRC32, [19:16]

Indicates whether CRC32 instructions are implemented. The value is:

0x1 CRC32 instructions are implemented.

SHA2, [15:12]

Indicates whether SHA2 instructions are implemented. The possible values are:

0x0 No SHA2 instructions are implemented. This is the value if the core implementation does not include the Cryptographic Extension.

0x1 SHA256H, SHA256H2, SHA256U0, and SHA256U1 implemented. This is the value if the core implementation includes the Cryptographic Extension.

SHA1, [11:8]

Indicates whether SHA1 instructions are implemented. The possible values are:

`0x0` No SHA1 instructions implemented. This is the value if the core implementation does not include the Cryptographic Extension.

`0x1` `SHA1C`, `SHA1P`, `SHA1M`, `SHA1SU0`, and `SHA1SU1` implemented. This is the value if the core implementation includes the Cryptographic Extension.

AES, [7:4]

Indicates whether AES instructions are implemented. The possible values are:

`0x0` No AES instructions implemented. This is the value if the core implementation does not include the Cryptographic Extension.

`0x2` `AESE`, `AESD`, `AESMC`, and `AESIMC` implemented, plus `PMULL` and `PMULL2` instructions operating on 64-bit data. This is the value if the core implementation includes the Cryptographic Extension.

[3:0]

Reserved, RES0.

Configurations

ID_AA64ISAR0_EL1 is architecturally mapped to external register ID_AA64ISAR0.

Bit fields and details that are not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

B2.57 ID_AA64ISAR1_EL1, AArch64 Instruction Set Attribute Register 1, EL1

The ID_AA64ISAR1_EL1 provides information about the instructions implemented in AArch64 state.

Bit field descriptions

ID_AA64ISAR1_EL1 is a 64-bit register, and is part of the Identification registers functional group.

This register is Read Only.

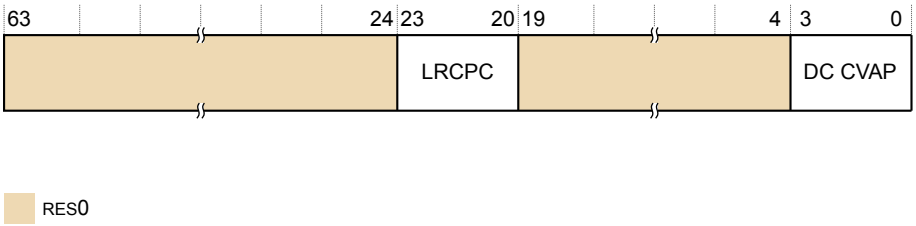

Figure B2-41 ID_AA64ISAR1_EL1 bit assignments

RES0, [63:24]

RES0 Reserved.

LRCPC, [23:20]

Indicates whether load-acquire (LDA) instructions are implemented for a Release Consistent core consistent RCPC model.

0x1 The LDAPRB, LDAPRH, and LDAPR instructions are implemented in AArch64.

RES0, [19:4]

RES0 Reserved.

DC CVAP, [3:0]

Indicates whether Data Cache, Clean to the Point of Persistence (DC CVAP) instructions are implemented.

0x1 DC CVAP is supported in AArch64.

Configurations

There are no configuration notes.

Bit fields and details that are not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

B2.58 ID_AA64MMFR0_EL1, AArch64 Memory Model Feature Register 0, EL1

The ID_AA64MMFR0_EL1 provides information about the implemented memory model and memory management support in the AArch64 Execution state.

Bit field descriptions

ID_AA64MMFR0_EL1 is a 64-bit register, and is part of the Identification registers functional group.

This register is Read Only.

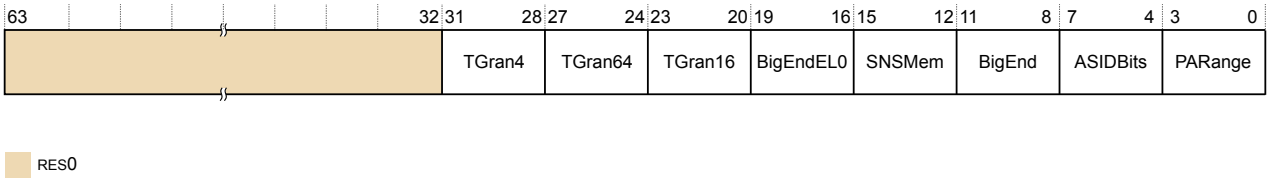

Figure B2-42 ID_AA64MMFR0_EL1 bit assignments

RES0, [63:32]

RES0 Reserved.

TGran4, [31:28]

Support for 4KB memory translation granule size:

0x0 4KB granule supported.

TGran64, [27:24]

Support for 64KB memory translation granule size:

0x0 64KB granule supported.

TGran16, [23:20]

Support for 16KB memory translation granule size:

0x1 Indicates that the 16KB granule is supported.

BigEndEL0, [19:16]

Mixed-endian support only at EL0.

0x0 No mixed-endian support at EL0. The SCTLR_EL1.E0E bit has a fixed value.

SNSMem, [15:12]

Secure versus Non-secure Memory distinction:

0x1 Supports a distinction between Secure and Non-secure Memory.

BigEnd, [11:8]

Mixed-endian configuration support:

0x1 Mixed-endian support. The SCTLR_ELx.EE and SCTLR_EL1.E0E bits can be configured.

ASIDBits, [7:4]

Number of ASID bits:

0x2 16 bits.

PARange, [3:0]

Physical address range supported:

`0x4` 44 bits, 16TB.

The supported Physical Address Range is 44 bits. Other cores in the DSU might support a different Physical Address Range.

Configurations

There are no configuration notes.

Bit fields and details that are not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

B2.59 ID_AA64MMFR1_EL1, AArch64 Memory Model Feature Register 1, EL1

The ID_AA64MMFR1_EL1 provides information about the implemented memory model and memory management support in the AArch64 Execution state.

Bit field descriptions

ID_AA64MMFR1_EL1 is a 64-bit register, and is part of the Identification registers functional group.

This register is Read Only.

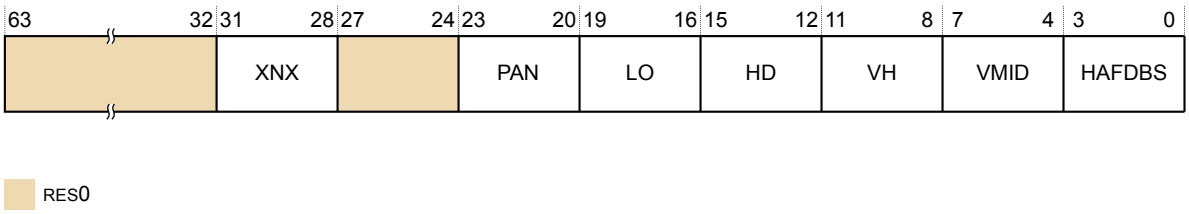

Figure B2-43 ID_AA64MMFR1_EL1 bit assignments

RES0, [63:32]

RES0 Reserved.

XNX, [31:28]

Indicates whether provision of EL0 vs EL1 execute never control at Stage 2 is supported.

0x1 EL0/EL1 execute control distinction at Stage 2 bit is supported. All other values are reserved.

RES0, [27:24]

RES0 Reserved.

PAN, [23:20]

Privileged Access Never. Indicates support for the PAN bit in PSTATE, SPSR_EL1, SPSR_EL2, SPSR_EL3, and DSPSR_EL0.

0x2 PAN supported and AT S1E1RP and AT S1E1WP instructions supported.

LO, [19:16]

Indicates support for LORegions.

0x1 LORegions are supported.

HD, [15:12]

Presence of Hierarchical Disables. Enables an operating system or hypervisor to hand over up to 4 bits of the last level page table descriptor (bits[62:59] of the page table entry) for use by hardware for IMPLEMENTATION DEFINED usage. The value is:

0x2 Hierarchical Permission Disables and Hardware allocation of bits[62:59] supported.

VH, [11:8]

Indicates whether Virtualization Host Extensions are supported.

0x1 Virtualization Host Extensions supported.

VMID, [7:4]

Indicates the number of VMID bits supported.

0x2 16 bits are supported.

HAFDBS, [3:0]

Indicates the support for hardware updates to Access flag and Dirty state in translation tables.

`0x2` Hardware update of both the Access flag and dirty state is supported in hardware.

Configurations

There are no configuration notes.

Bit fields and details that are not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

B2.60 ID_AA64MMFR2_EL1, AArch64 Memory Model Feature Register 2, EL1

The ID_AA64MMFR2_EL1 provides information about the implemented memory model and memory management support in the AArch64 Execution state.

Bit field descriptions

ID_AA64MMFR2_EL1 is a 64-bit register, and is part of the Identification registers functional group.

This register is Read Only.

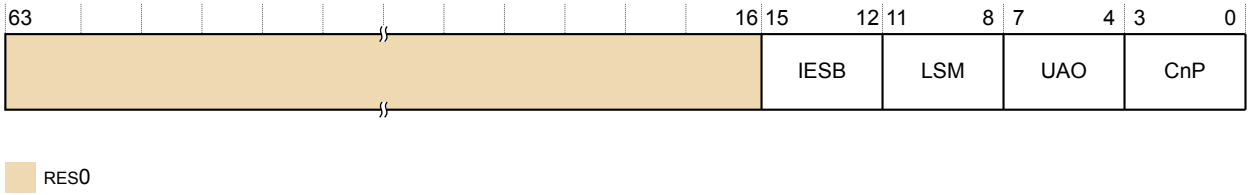

Figure B2-44 ID_AA64MMFR2_EL1 bit assignments

RES0, [63:16]

RES0 Reserved.

IESB, [15:12]

Indicates whether an implicit Error Synchronization Barrier has been inserted. The value is:

`0x1` SCTLR_ELx.IESB implicit `ErrorSynchronizationBarrier` control implemented.

LSM, [11:8]

Indicates whether LDM and STM ordering control bits are supported. The value is:

`0x0` LSMAOE and nTLSMD bit not supported.

UAO, [7:4]

Indicates the presence of the *User Access Override* (UAO). The value is:

`0x1` UAO is supported.

CnP, [3:0]

Common not Private. Indicates whether a TLB entry is pointed at a translation table base register that is a member of a common set. The value is:

`0x1` CnP bit is supported.

Configurations

There are no configuration notes.

Bit fields and details that are not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

B2.61 ID_AA64PFR0_EL1, AArch64 Processor Feature Register 0, EL1

The ID_AA64PFR0_EL1 provides additional information about implemented core features in AArch64.

The optional Advanced SIMD and floating-point support is not included in the base product of the core. Arm requires licensees to have contractual rights to obtain the Advanced SIMD and floating-point support.

Bit field descriptions

ID_AA64PFR0_EL1 is a 64-bit register, and is part of the Identification registers functional group.

This register is Read Only.

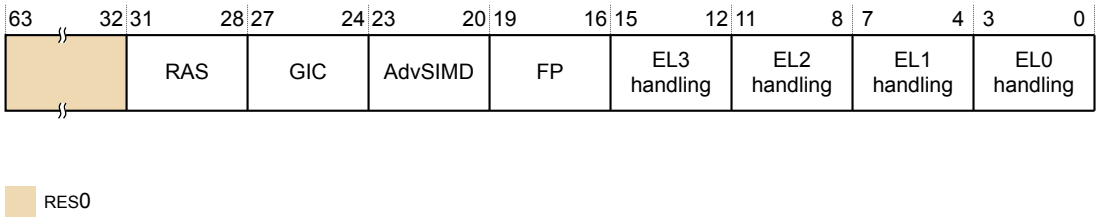

Figure B2-45 ID_AA64PFR0_EL1 bit assignments

RES0, [63:32]

RES0 Reserved.

RAS, [31:28]

RAS extension version. The possible values are:

0x1 Version 1 of the RAS extension is present.

GIC, [27:24]

GIC CPU interface:

0x0 GIC CPU interface is disabled, GICCDISABLE is HIGH, or not implemented.

0x1 GIC CPU interface is implemented and enabled, GICCDISABLE is LOW.

AdvSIMD, [23:20]

Advanced SIMD. The possible values are:

0x1 Advanced SIMD, including Half-precision support, is implemented.

FP, [19:16]

Floating-point. The possible values are:

0x1 Floating-point, including Half-precision support, is implemented.

EL3 handling, [15:12]

EL3 exception handling:

0x2 Instructions can be executed at EL3 in AArch64 or AArch32 state.

EL2 handling, [11:8]

EL2 exception handling:

0x2 Instructions can be executed at EL3 in AArch64 or AArch32 state.

EL1 handling, [7:4]

EL1 exception handling. The possible values are:

0x2 Instructions can be executed at EL3 in AArch64 or AArch32 state.

EL0 handling, [3:0]

EL0 exception handling. The possible values are:

`0x2` Instructions can be executed at EL0 in AArch64 or AArch32 state.

Configurations

ID_AA64PFR0_EL1 is architecturally mapped to External register EDPFR.

Bit fields and details that are not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

B2.62 ID_AFR0_EL1, AArch32 Auxiliary Feature Register 0, EL1

The ID_AFR0_EL1 provides information about the IMPLEMENTATION DEFINED features of the PE in AArch32. This register is not used in the Cortex-A75 core.

Bit field descriptions

ID_AFR0_EL1 is a 32-bit register, and is part of the Identification registers functional group.

This register is Read Only.

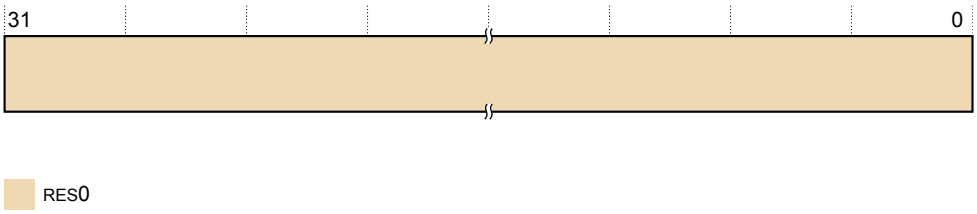

Figure B2-46 ID_AFR0_EL1 bit assignments

RES0, [31:0]

Reserved, RES0.

Configurations

AArch64 System register ID_AFR0_EL1 is architecturally mapped to AArch32 System register ID_AFR0. See *B1.57 ID_AFR0, Auxiliary Feature Register 0* on page B1-211.

Bit fields and details that are not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

B2.63 ID_DFR0_EL1, AArch32 Debug Feature Register 0, EL1

The ID_DFR0_EL1 provides top-level information about the debug system in AArch32.

Bit field descriptions

ID_DFR0_EL1 is a 32-bit register, and is part of the Identification registers functional group.

This register is Read Only.

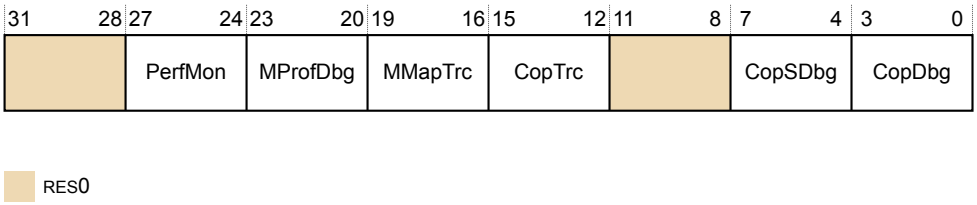

Figure B2-47 ID_DFR0_EL1 bit assignments

RES0, [31:28]

RES0 Reserved.

PerfMon, [27:24]

Indicates support for performance monitor model:

`4` Support for *Performance Monitor Unit version 3* (PMUv3) system registers, with a 16-bit evtCount field.

MProfDbg, [23:20]

Indicates support for memory-mapped debug model for M profile cores:

`0` This product does not support M profile Debug architecture.

MMapTrc, [19:16]

Indicates support for memory-mapped trace model:

`1` Support for Arm trace architecture, with memory-mapped access.

In the Trace registers, the ETMIDR gives more information about the implementation.

CopTrc, [15:12]

Indicates support for coprocessor-based trace model:

`0` This product does not support Arm trace architecture.

RES0, [11:8]

RES0 Reserved.

CopSDbg, [7:4]

Indicates support for coprocessor-based Secure debug model:

`8` This product supports v8.2 Debug architecture.

CopDbg, [3:0]

Indicates support for coprocessor-based debug model:

`8` This product supports v8.2 Debug architecture.

Configurations

ID_DFR0_EL1 is architecturally mapped to AArch32 register ID_DFR0. See *B1.58 ID_DFR0, Debug Feature Register 0* on page B1-212.

Bit fields and details that are not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

B2.64 ID_ISAR0_EL1, AArch32 Instruction Set Attribute Register 0, EL1

The ID_ISAR0_EL1 provides information about the instruction sets implemented by the core in AArch32.

Bit field descriptions

ID_ISAR0_EL1 is a 32-bit register, and is part of the Identification registers functional group.

This register is Read Only.

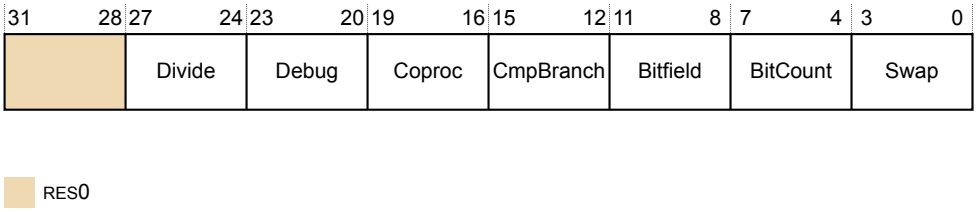

Figure B2-48 ID_ISAR0_EL1 bit assignments

RES0, [31:28]

RES0 Reserved.

Divide, [27:24]

Indicates the implemented Divide instructions:

0x2
- SDIV and UDIV in the T32 instruction set.
- SDIV and UDIV in the A32 instruction set.

Debug, [23:20]

Indicates the implemented Debug instructions:

0x1 BKPT.

Coproc, [19:16]

Indicates the implemented Coprocessor instructions:

0x0 None implemented, except for instructions separately attributed by the architecture to provide access to AArch32 System registers and System instructions.

CmpBranch, [15:12]

Indicates the implemented combined Compare and Branch instructions in the T32 instruction set:

0x1 CBNZ and CBZ.

Bitfield, [11:8]

Indicates the implemented bit field instructions:

0x1 BFC, BFI, SBFX, and UBFX.

BitCount, [7:4]

Indicates the implemented Bit Counting instructions:

0x1 CLZ.

Swap, [3:0]

Indicates the implemented Swap instructions in the A32 instruction set:

0x0 None implemented.

Configurations

ID_ISAR0_EL1 is architecturally mapped to AArch32 register ID_ISAR0. See *B1.59 ID_ISAR0, Instruction Set Attribute Register 0* on page B1-214.

In an AArch64-only implementation, this register is UNKNOWN.

Must be interpreted with ID_ISAR1_EL1, ID_ISAR2_EL1, ID_ISAR3_EL1, ID_ISAR4_EL1, and ID_ISAR5_EL1. See:

- *B2.65 ID_ISAR1_EL1, AArch32 Instruction Set Attribute Register 1, EL1* on page B2-378.
- *B2.66 ID_ISAR2_EL1, AArch32 Instruction Set Attribute Register 2, EL1* on page B2-380.
- *B2.67 ID_ISAR3_EL1, AArch32 Instruction Set Attribute Register 3, EL1* on page B2-382.
- *B2.68 ID_ISAR4_EL1, AArch32 Instruction Set Attribute Register 4, EL1* on page B2-384.
- *B2.69 ID_ISAR5_EL1, AArch32 Instruction Set Attribute Register 5, EL1* on page B2-386.

Bit fields and details that are not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

B2.65 ID_ISAR1_EL1, AArch32 Instruction Set Attribute Register 1, EL1

The ID_ISAR1_EL1 provides information about the instruction sets implemented by the core in AArch32.

Bit field descriptions

ID_ISAR1_EL1 is a 32-bit register, and is part of the Identification registers functional group.

This register is Read Only.

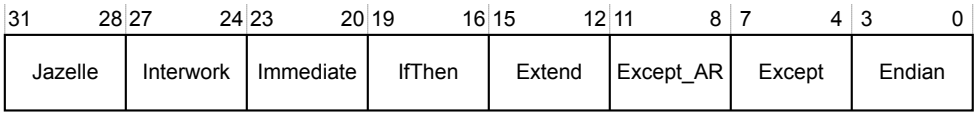

Figure B2-49 ID_ISAR1_EL1 bit assignments

Jazelle, [31:28]
Indicates the implemented Jazelle state instructions:

`0x1` Adds the `BXJ` instruction, and the J bit in the PSR.

Interwork, [27:24]
Indicates the implemented Interworking instructions:

`0x3`
- The `BX` instruction, and the T bit in the PSR.
- The `BLX` instruction. The PC loads have `BX`-like behavior.
- Data-processing instructions in the A32 instruction set with the PC as the destination and the S bit clear, have `BX`-like behavior.

Immediate, [23:20]
Indicates the implemented data-processing instructions with long immediates:

`0x1`
- The `MOVT` instruction.
- The `MOV` instruction encodings with zero-extended 16-bit immediates.
- The T32 `ADD` and `SUB` instruction encodings with zero-extended 12-bit immediates, and other `ADD`, `ADR`, and `SUB` encodings cross-referenced by the pseudocode for those encodings.

IfThen, [19:16]
Indicates the implemented If-Then instructions in the T32 instruction set:

`0x1` The `IT` instructions, and the IT bits in the PSRs.

Extend, [15:12]
Indicates the implemented Extend instructions:

`0x2`
- The `SXTB`, `SXTH`, `UXTB`, and `UXTH` instructions.
- The `SXTB16`, `SXTAB`, `SXTAB16`, `SXTAH`, `UXTB16`, `UXTAB`, `UXTAB16`, and `UXTAH` instructions.

Except_AR, [11:8]
Indicates the implemented A profile exception-handling instructions:

`0x1` The `SRS` and `RFE` instructions, and the A profile forms of the `CPS` instruction.

Except, [7:4]
Indicates the implemented exception-handling instructions in the A32 instruction set:

`0x1` The `LDM` (exception return), `LDM` (user registers), and `STM` (user registers) instruction versions.

Endian, [3:0]
Indicates the implemented Endian instructions:

`0x1` The `SETEND` instruction, and the E bit in the PSRs.

Configurations

ID_ISAR1_EL1 is architecturally mapped to AArch32 register ID_ISAR1. See *B1.60 ID_ISAR1, Instruction Set Attribute Register 1* on page B1-216.

In an AArch64-only implementation, this register is UNKNOWN.

Must be interpreted with ID_ISAR0_EL1, ID_ISAR2_EL1, ID_ISAR3_EL1, ID_ISAR4_EL1 and ID_ISAR5_EL1. See:

- *B2.64 ID_ISAR0_EL1, AArch32 Instruction Set Attribute Register 0, EL1* on page B2-376.
- *B2.66 ID_ISAR2_EL1, AArch32 Instruction Set Attribute Register 2, EL1* on page B2-380.
- *B2.67 ID_ISAR3_EL1, AArch32 Instruction Set Attribute Register 3, EL1* on page B2-382.
- *B2.68 ID_ISAR4_EL1, AArch32 Instruction Set Attribute Register 4, EL1* on page B2-384.
- *B2.69 ID_ISAR5_EL1, AArch32 Instruction Set Attribute Register 5, EL1* on page B2-386.

Bit fields and details that are not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

B2.66 ID_ISAR2_EL1, AArch32 Instruction Set Attribute Register 2, EL1

The ID_ISAR2_EL1 provides information about the instruction sets implemented by the core in AArch32.

Bit field descriptions

ID_ISAR2_EL1 is a 32-bit register, and is part of the Identification registers functional group.

This register is Read Only.

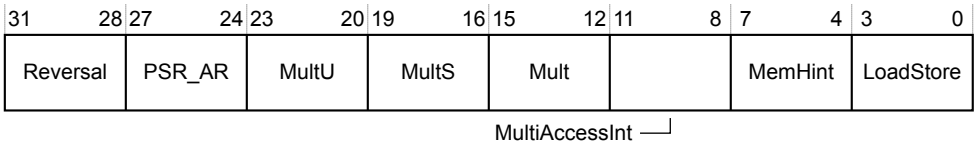

Figure B2-50 ID_ISAR2_EL1 bit assignments

Reversal, [31:28]

Indicates the implemented Reversal instructions:

0x2 The REV, REV16, REVSH, and RBIT instructions.

PSR_AR, [27:24]

Indicates the implemented A and R profile instructions to manipulate the PSR:

0x1 The MRS and MSR instructions, and the exception return forms of data-processing instructions.

The exception return forms of the data-processing instructions are:

- In the A32 instruction set, data-processing instructions with the PC as the destination and the S bit set.
- In the T32 instruction set, the SUBSPC, LR, #N instruction.

MultU, [23:20]

Indicates the implemented advanced unsigned Multiply instructions:

0x2 The UMULL, UMLAL, and UMAAL instructions.

MultS, [19:16]

Indicates the implemented advanced signed Multiply instructions.

0x3
- The SMULL and SMLAL instructions.
- The SMLABB, SMLABT, SMLALBB, SMLALBT, SMLALTB, SMLALTT, SMLATB, SMLATT, SMLAWB, SMLAWT, SMULBB, SMULBT, SMULTB, SMULTT, SMULWB, SMULWT instructions, and the Q bit in the PSRs.
- The SMLAD, SMLADX, SMLALD, SMLALDX, SMLSD, SMLSDX, SMLSLD, SMLSLDX, SMMLA, SMMLAR, SMMLS, SMMLSR, SMMUL, SMMULR, SMUAD, SMUADX, SMUSD, and SMUSDX instructions.

Mult, [15:12]

Indicates the implemented additional Multiply instructions:

0x2 The MUL, MLA and MLS instructions.

MultiAccessInt, [11:8]

Indicates the support for interruptible multi-access instructions:

0x0 No support. This means the LDM and STM instructions are not interruptible.

MemHint, [7:4]

Indicates the implemented memory hint instructions:

0x4 The PLD, PLI, and PLDWinstructions.

LoadStore, [3:0]

Indicates the implemented additional load/store instructions:

`0x2` The `LDRD` and `STRD` instructions.

The Load Acquire (`LDAB`, `LDAH`, `LDA`, `LDAEXB`, `LDAEXH`, `LDAEX`, and `LDAEXD`) and Store Release (`STLB`, `STLH`, `STL`, `STLEXB`, `STLEXH`, `STLEX`, and `STLEXD`) instructions.

Configurations

ID_ISAR2_EL1 is architecturally mapped to AArch32 register ID_ISAR2. See *B1.61 ID_ISAR2, Instruction Set Attribute Register 2* on page B1-218.

In an AArch64-only implementation, this register is UNKNOWN.

Must be interpreted with ID_ISAR0_EL1, ID_ISAR1_EL1, ID_ISAR3_EL1, ID_ISAR4_EL1, and ID_ISAR5_EL1. See:

- *B2.64 ID_ISAR0_EL1, AArch32 Instruction Set Attribute Register 0, EL1* on page B2-376.
- *B2.65 ID_ISAR1_EL1, AArch32 Instruction Set Attribute Register 1, EL1* on page B2-378.
- *B2.67 ID_ISAR3_EL1, AArch32 Instruction Set Attribute Register 3, EL1* on page B2-382.
- *B2.68 ID_ISAR4_EL1, AArch32 Instruction Set Attribute Register 4, EL1* on page B2-384.
- *B2.69 ID_ISAR5_EL1, AArch32 Instruction Set Attribute Register 5, EL1* on page B2-386.

Bit fields and details that are not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

B2.67 ID_ISAR3_EL1, AArch32 Instruction Set Attribute Register 3, EL1

The ID_ISAR3_EL1 provides information about the instruction sets implemented by the core in AArch32.

Bit field descriptions

ID_ISAR3_EL1 is a 32-bit register, and is part of the Identification registers functional group.

This register is Read Only.

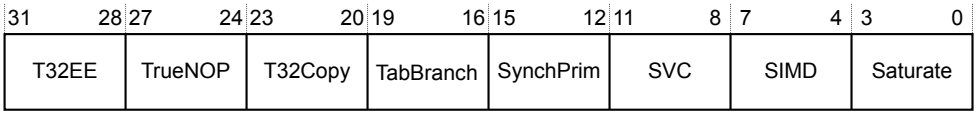

Figure B2-51 ID_ISAR3_EL1 bit assignments

T32EE, [31:28]
Indicates the implemented T32EE instructions:

`0x0` None implemented.

TrueNOP, [27:24]
Indicates support for True NOP instructions:

`0x1` True `NOP` instructions in both the A32 and T32 instruction sets, and additional NOP-compatible hints.

T32Copy, [23:20]
Indicates the support for T32 non flag-setting `MOV` instructions:

`0x1` Support for T32 instruction set encoding T1 of the `MOV` (register) instruction, copying from a low register to a low register.

TabBranch, [19:16]
Indicates the implemented Table Branch instructions in the T32 instruction set.

`0x1` The `TBB` and `TBH` instructions.

SynchPrim, [15:12]
Indicates the implemented Synchronization Primitive instructions:

`0x2`
- The `LDREX` and `STREX` instructions.
- The `CLREX`, `LDREXB`, `STREXB`, and `STREXH` instructions.
- The `LDREXD` and `STREXD` instructions.

SVC, [11:8]
Indicates the implemented SVC instructions:

`0x1` The `SVC` instruction.

SIMD, [7:4]
Indicates the implemented *Single Instruction Multiple Data* (SIMD) instructions.

`0x3` • The `SSAT` and `USAT` instructions, and the Q bit in the PSRs.
• The `PKHBT`, `PKHTB`, `QADD16`, `QADD8`, `QASX`, `QSUB16`, `QSUB8`, `QSAX`, `SADD16`, `SADD8`, `SASX`, `SEL`, `SHADD16`, `SHADD8`, `SHASX`, `SHSUB16`, `SHSUB8`, `SHSAX`, `SSAT16`, `SSUB16`, `SSUB8`, `SSAX`, `SXTAB16`, `SXTB16`, `UADD16`, `UADD8`, `UASX`, `UHADD16`, `UHADD8`, `UHASX`, `UHSUB16`, `UHSUB8`, `UHSAX`, `UQADD16`, `UQADD8`, `UQASX`, `UQSUB16`, `UQSUB8`, `UQSAX`, `USAD8`, `USADA8`, `USAT16`, `USUB16`, `USUB8`, `USAX`, `UXTAB16`, `UXTB16` instructions, and the GE[3:0] bits in the PSRs.

The SIMD field relates only to implemented instructions that perform SIMD operations on the general-purpose registers. In an implementation that supports Advanced SIMD and floating-point instructions, MVFR0, MVFR1, and MVFR2 give information about the implemented Advanced SIMD instructions.

Saturate, [3:0]

Indicates the implemented Saturate instructions:

`0x1` The `QADD`, `QDADD`, `QDSUB`, `QSUB` Q bit in the PSRs.

Configurations

ID_ISAR3_EL1 is architecturally mapped to AArch32 register ID_ISAR3. See *B1.62 ID_ISAR3, Instruction Set Attribute Register 3* on page B1-220.

In an AArch64-only implementation, this register is UNKNOWN.

Must be interpreted with ID_ISAR0_EL1, ID_ISAR1_EL1, ID_ISAR2_EL1, ID_ISAR4_EL1, and ID_ISAR5_EL1. See:

- *B2.64 ID_ISAR0_EL1, AArch32 Instruction Set Attribute Register 0, EL1* on page B2-376.
- *B2.65 ID_ISAR1_EL1, AArch32 Instruction Set Attribute Register 1, EL1* on page B2-378.
- *B2.66 ID_ISAR2_EL1, AArch32 Instruction Set Attribute Register 2, EL1* on page B2-380.
- *B2.68 ID_ISAR4_EL1, AArch32 Instruction Set Attribute Register 4, EL1* on page B2-384.
- *B2.69 ID_ISAR5_EL1, AArch32 Instruction Set Attribute Register 5, EL1* on page B2-386.

Bit fields and details that are not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

B2.68 ID_ISAR4_EL1, AArch32 Instruction Set Attribute Register 4, EL1

The ID_ISAR4_EL1 provides information about the instruction sets implemented by the core in AArch32.

Bit field descriptions

ID_ISAR4_EL1 is a 32-bit register, and is part of the Identification registers functional group.

This register is Read Only.

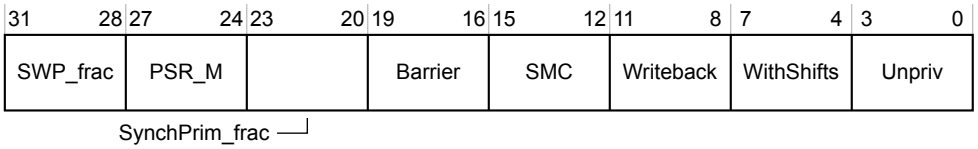

Figure B2-52 ID_ISAR4_EL1 bit assignments

SWP_frac, [31:28]
Indicates support for the memory system locking the bus for `SWP` or `SWPB` instructions:

`0x0` `SWP` and `SWPB` instructions not implemented.

PSR_M, [27:24]
Indicates the implemented M profile instructions to modify the PSRs:

`0x0` None implemented.

SynchPrim_frac, [23:20]
This field is used with the ID_ISAR3.SynchPrim field to indicate the implemented Synchronization Primitive instructions:

`0x0`
- The `LDREX` and `STREX` instructions.
- The `CLREX`, `LDREXB`, `LDREXH`, `STREXB`, and `STREXH` instructions.
- The `LDREXD` and `STREXD` instructions.

Barrier, [19:16]
Indicates the supported Barrier instructions in the A32 and T32 instruction sets:

`0x1` The `DMB`, `DSB`, and `ISB` barrier instructions.

SMC, [15:12]
Indicates the implemented `SMC` instructions:

`0x1` The `SMC` instruction.

WriteBack, [11:8]
Indicates the support for Write-Back addressing modes:

`0x1` Core supports all the Write-Back addressing modes as defined in Armv8.

WithShifts, [7:4]
Indicates the support for instructions with shifts.

`0x4`
- Support for shifts of loads and stores over the range LSL 0-3.
- Support for other constant shift options, both on load/store and other instructions.
- Support for register-controlled shift options.

Unpriv, [3:0]
Indicates the implemented unprivileged instructions.

`0x2`
- The `LDRBT`, `LDRT`, `STRBT`, and `STRT` instructions.
- The `LDRHT`, `LDRSBT`, `LDRSHT`, and `STRHT` instructions.

Configurations

ID_ISAR4_EL1 is architecturally mapped to AArch32 register ID_ISAR4. See *B1.63 ID_ISAR4, Instruction Set Attribute Register 4* on page B1-222.

In an AArch64-only implementation, this register is UNKNOWN.

Must be interpreted with ID_ISAR0_EL1, ID_ISAR1_EL1, ID_ISAR2_EL1, ID_ISAR3_EL1, and ID_ISAR5_EL1. See:

- *B2.64 ID_ISAR0_EL1, AArch32 Instruction Set Attribute Register 0, EL1* on page B2-376.
- *B2.65 ID_ISAR1_EL1, AArch32 Instruction Set Attribute Register 1, EL1* on page B2-378.
- *B2.66 ID_ISAR2_EL1, AArch32 Instruction Set Attribute Register 2, EL1* on page B2-380.
- *B2.67 ID_ISAR3_EL1, AArch32 Instruction Set Attribute Register 3, EL1* on page B2-382.
- *B2.69 ID_ISAR5_EL1, AArch32 Instruction Set Attribute Register 5, EL1* on page B2-386.

Bit fields and details that are not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

B2.69 ID_ISAR5_EL1, AArch32 Instruction Set Attribute Register 5, EL1

The ID_ISAR5_EL1 provides information about the instruction sets that the core implements.

Bit field descriptions

ID_ISAR5_EL1 is a 32-bit register, and is part of the Identification registers functional group.

This register is Read Only.

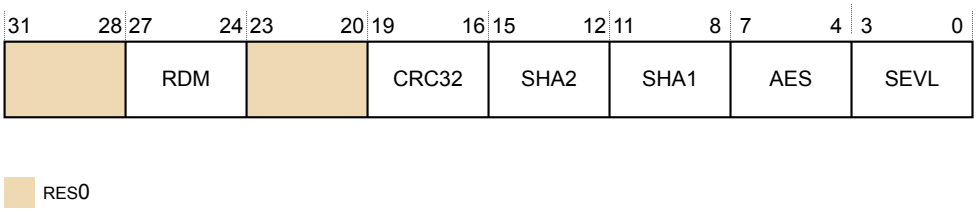

Figure B2-53 ID_ISAR5_EL1 bit assignments

[31:28]

RES0 Reserved.

RDM, [27:24]

VQRDMLAH and VQRDMLSH instructions in AArch32. The value is:

0x1 VQRDMLAH and VQRDMLSH instructions are implemented.

[23:20]

RES0 Reserved.

CRC32, [19:16]

Indicates whether CRC32 instructions are implemented in AArch32 state. The value is:

0x1 CRC32B, CRC32H, CRC32W, CRC32CB, CRC32CH, and CRC32CW instructions are implemented.

SHA2, [15:12]

Indicates whether SHA2 instructions are implemented in AArch32 state. The possible values are:

0x0 No SHA2 instructions implemented. This is the value when the Cryptographic Extensions are not implemented or are disabled.

0x1 SHA256H, SHA256H2, SHA256SU0, and SHA256SU1 instructions are implemented. This is the value when the Cryptographic Extensions are implemented and enabled.

SHA1, [11:8]

Indicates whether SHA1 instructions are implemented in AArch32 state. The possible values are:

0x0 No SHA1 instructions implemented. This is the value when the Cryptographic Extensions are not implemented or are disabled.

0x1 SHA1C, SHA1P, SHA1M, SHA1H, SHA1SU0, and SHA1SU1 instructions are implemented. This is the value when the Cryptographic Extensions are implemented and enabled.

AES, [7:4]

Indicates whether AES instructions are implemented in AArch32 state. The possible values are:

0x0 No AES instructions implemented. This is the value when the Cryptographic Extensions are not implemented or are disabled.

`0x2` • `AESE`, `AESD`, `AESMC`, and `AESIMC` implemented.
• `PMULL` and `PMULL2` instructions operating on 64-bit data.

This is the value when the Cryptographic Extensions are implemented and enabled.

SEVL, [3:0]

Indicates whether the `SEVL` instruction is implemented:

`0x1` `SEVL` implemented to send event local.

Configurations

ID_ISAR5_EL1 is architecturally mapped to AArch32 register ID_ISAR5. See *B1.64 ID_ISAR5, Instruction Set Attribute Register 5* on page B1-224.
ID_ISAR5 must be interpreted with ID_ISAR0_EL1, ID_ISAR1_EL1, ID_ISAR2_EL1, ID_ISAR3_EL1, and ID_ISAR4_EL1. See:

- *B2.64 ID_ISAR0_EL1, AArch32 Instruction Set Attribute Register 0, EL1* on page B2-376.
- *B2.65 ID_ISAR1_EL1, AArch32 Instruction Set Attribute Register 1, EL1* on page B2-378.
- *B2.66 ID_ISAR2_EL1, AArch32 Instruction Set Attribute Register 2, EL1* on page B2-380.
- *B2.67 ID_ISAR3_EL1, AArch32 Instruction Set Attribute Register 3, EL1* on page B2-382.
- *B2.68 ID_ISAR4_EL1, AArch32 Instruction Set Attribute Register 4, EL1* on page B2-384.

Bit fields and details that are not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

B2.70 ID_ISAR6_EL1, AArch32 Instruction Set Attribute Register 6, EL1

The ID_ISAR6 provides information about the instruction sets that the core implements.

Bit field descriptions

ID_ISAR6_EL1 is a 32-bit register, and is part of the Identification registers functional group.

This register is Read Only.

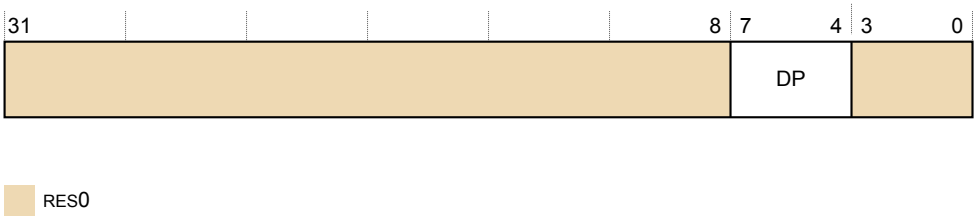

Figure B2-54 ID_ISAR6_EL1 bit assignments

RES0, [31:8]

RES0 Reserved.

DP, [7:4]

UDOT and SDOT instructions. The value is:

0b0001 UDOT and SDOT instructions are implemented.

RES0, [3:0]

RES0 Reserved.

Configurations

ID_ISAR6_EL1 is architecturally mapped to AArch32 register ID_ISAR6. See *B1.65 ID_ISAR6, Instruction Set Attribute Register 6* on page B1-226.

There is one copy of this register that is used in both Secure and Non-secure states.

ID_ISAR6_EL1 must be interpreted with ID_ISAR0_EL1, ID_ISAR1_EL1, ID_ISAR2_EL1, ID_ISAR3_EL1, ID_ISAR4_EL1, and ID_ISAR5_EL1. See:

- *B2.64 ID_ISAR0_EL1, AArch32 Instruction Set Attribute Register 0, EL1* on page B2-376.
- *B2.65 ID_ISAR1_EL1, AArch32 Instruction Set Attribute Register 1, EL1* on page B2-378.
- *B2.66 ID_ISAR2_EL1, AArch32 Instruction Set Attribute Register 2, EL1* on page B2-380.
- *B2.67 ID_ISAR3_EL1, AArch32 Instruction Set Attribute Register 3, EL1* on page B2-382.
- *B2.68 ID_ISAR4_EL1, AArch32 Instruction Set Attribute Register 4, EL1* on page B2-384.
- *B2.69 ID_ISAR5_EL1, AArch32 Instruction Set Attribute Register 5, EL1* on page B2-386.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

B2.71 ID_MMFR0_EL1, AArch32 Memory Model Feature Register 0, EL1

The ID_MMFR0_EL1 provides information about the memory model and memory management support in AArch32.

Bit field descriptions

ID_MMFR0_EL1 is a 32-bit register, and is part of the Identification registers functional group.

This register is Read Only.

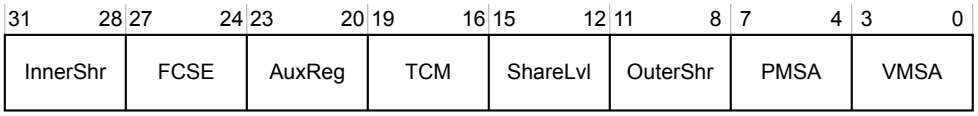

Figure B2-55 ID_MMFR0_EL1 bit assignments

InnerShr, [31:28]
Indicates the innermost shareability domain implemented:

`0x1` Implemented with hardware coherency support.

FCSE, [27:24]
Indicates support for *Fast Context Switch Extension* (FCSE):

`0x0` Not supported.

AuxReg, [23:20]
Indicates support for Auxiliary registers:

`0x2` Support for Auxiliary Fault Status Registers (AIFSR and ADFSR) and Auxiliary Control Register.

TCM, [19:16]
Indicates support for TCMs and associated DMAs:

`0x0` Not supported.

ShareLvl, [15:12]
Indicates the number of shareability levels implemented:

`0x1` Two levels of shareability implemented.

OuterShr, [11:8]
Indicates the outermost shareability domain implemented:

`0x1` Implemented with hardware coherency support.

PMSA, [7:4]
Indicates support for a *Protected Memory System Architecture* (PMSA):

`0x0` Not supported.

VMSA, [3:0]
Indicates support for a *Virtual Memory System Architecture* (VMSA).

`0x5` Support for:
- VMSAv7, with support for remapping and the Access flag.
- The PXN bit in the Short-descriptor translation table format descriptors.
- The Long-descriptor translation table format.

Configurations

ID_MMFR0_EL1 is architecturally mapped to AArch32 register ID_MMFR0. See *B1.66 ID_MMFR0, Memory Model Feature Register 0* on page B1-227.

Must be interpreted with ID_MMFR1_EL1, ID_MMFR2_EL1, ID_MMFR3_EL1, and ID_MMFR4_EL1. See:

- *B2.72 ID_MMFR1_EL1, AArch32 Memory Model Feature Register 1, EL1* on page B2-391.
- *B2.73 ID_MMFR2_EL1, AArch32 Memory Model Feature Register 2, EL1* on page B2-393.
- *B2.74 ID_MMFR3_EL1, AArch32 Memory Model Feature Register 3, EL1* on page B2-395.
- *B2.75 ID_MMFR4_EL1, AArch32 Memory Model Feature Register 4, EL1* on page B2-397.

Bit fields and details that are not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

B2.72 ID_MMFR1_EL1, AArch32 Memory Model Feature Register 1, EL1

The ID_MMFR1_EL1 provides information about the memory model and memory management support in AArch32.

Bit field descriptions

ID_MMFR1_EL1 is a 32-bit register, and is part of the Identification registers functional group.

This register is Read Only.

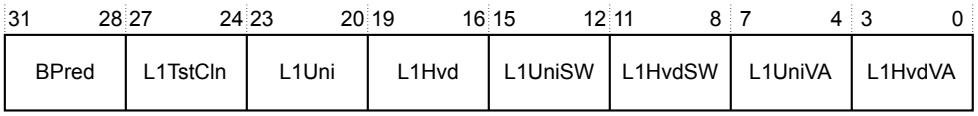

Figure B2-56 ID_MMFR1_EL1 bit assignments

BPred, [31:28]
Indicates branch predictor management requirements:

`0x4` For execution correctness, branch predictor requires no flushing at any time.

L1TstCln, [27:24]
Indicates the supported L1 Data cache test and clean operations, for Harvard or unified cache implementation:

`0x0` None supported.

L1Uni, [23:20]
Indicates the supported entire L1 cache maintenance operations, for a unified cache implementation:

`0x0` None supported.

L1Hvd, [19:16]
Indicates the supported entire L1 cache maintenance operations, for a Harvard cache implementation:

`0x0` None supported.

L1UniSW, [15:12]
Indicates the supported L1 cache line maintenance operations by set/way, for a unified cache implementation:

`0x0` None supported.

L1HvdSW, [11:8]
Indicates the supported L1 cache line maintenance operations by set/way, for a Harvard cache implementation:

`0x0` None supported.

L1UniVA, [7:4]
Indicates the supported L1 cache line maintenance operations by MVA, for a unified cache implementation:

`0x0` None supported.

L1HvdVA, [3:0]
Indicates the supported L1 cache line maintenance operations by MVA, for a Harvard cache implementation:

`0x0` None supported.

Configurations

ID_MMFR1_EL1 is architecturally mapped to AArch32 register ID_MMFR1. See *B1.67 ID_MMFR1, Memory Model Feature Register 1* on page B1-229.

Must be interpreted with ID_MMFR0_EL1, ID_MMFR2_EL1, ID_MMFR3_EL1, and ID_MMFR4_EL1. See:

- *B2.71 ID_MMFR0_EL1, AArch32 Memory Model Feature Register 0, EL1* on page B2-389.
- *B2.73 ID_MMFR2_EL1, AArch32 Memory Model Feature Register 2, EL1* on page B2-393.
- *B2.74 ID_MMFR3_EL1, AArch32 Memory Model Feature Register 3, EL1* on page B2-395.
- *B2.75 ID_MMFR4_EL1, AArch32 Memory Model Feature Register 4, EL1* on page B2-397.

Bit fields and details that are not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

B2.73 ID_MMFR2_EL1, AArch32 Memory Model Feature Register 2, EL1

The ID_MMFR2_EL1 provides information about the implemented memory model and memory management support in AArch32.

Bit field descriptions

ID_MMFR2_EL1 is a 32-bit register, and is part of the Identification registers functional group.

This register is Read Only.

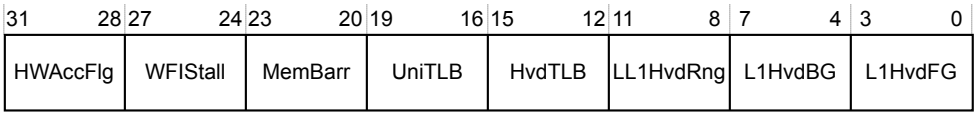

Figure B2-57 ID_MMFR2_EL1 bit assignments

HWAccFlg, [31:28]
Hardware access flag. Indicates support for a hardware access flag, as part of the VMSAv7 implementation:

`0x0` Not supported.

WFIStall, [27:24]
Wait For Interrupt Stall. Indicates the support for *Wait For Interrupt* (WFI) stalling:

`0x1` Support for WFI stalling.

MemBarr, [23:20]
Memory Barrier. Indicates the supported CP15 memory barrier operations.

`0x2` Supported CP15 memory barrier operations are:
- *Data Synchronization Barrier* (DSB).
- *Instruction Synchronization Barrier* (ISB).
- *Data Memory Barrier* (DMB).

UniTLB, [19:16]
Unified TLB. Indicates the supported TLB maintenance operations, for a unified TLB implementation.

`0x6` Supported unified TLB maintenance operations are:
- Invalidate all entries in the TLB.
- Invalidate TLB entry by MVA.
- Invalidate TLB entries by ASID match.
- Invalidate instruction TLB and data TLB entries by MVA All ASID. This is a shared unified TLB operation.
- Invalidate Hyp mode unified TLB entry by MVA.
- Invalidate entire Non-secure EL1 and EL0 unified TLB.
- Invalidate entire Hyp mode unified TLB.
- `TLBIMVALIS`, `TLBIMVAALIS`, `TLBIMVALHIS`, `TLBIMVAL`, `TLBIMVAAL`, and `TLBIMVALH`.
- `TLBIIPAS2IS`, `TLBIIPAS2LIS`, `TLBIIPAS2`, and `TLBIIPAS2L`.

HvdTLB, [15:12]
Harvard TLB. Indicates the supported TLB maintenance operations, for a Harvard TLB implementation:

`0x0` Not supported.

LL1HvdRng, [11:8]
L1 Harvard cache Range. Indicates the supported L1 cache maintenance range operations, for a Harvard cache implementation:

`0x0` Not supported.

L1HvdBG, [7:4]

L1 Harvard cache Background fetch. Indicates the supported L1 cache background prefetch operations, for a Harvard cache implementation:

`0x0` Not supported.

L1HvdFG, [3:0]

L1 Harvard cache Foreground fetch. Indicates the supported L1 cache foreground prefetch operations, for a Harvard cache implementation:

`0x0` Not supported.

Configurations

ID_MMFR2_EL1 is architecturally mapped to AArch32 register ID_MMFR2. See *B1.68 ID_MMFR2, Memory Model Feature Register 2* on page B1-231.

Must be interpreted with ID_MMFR0_EL1, ID_MMFR1_EL1, ID_MMFR3_EL1, and ID_MMFR4_EL1. See:

- *B2.71 ID_MMFR0_EL1, AArch32 Memory Model Feature Register 0, EL1* on page B2-389.
- *B2.72 ID_MMFR1_EL1, AArch32 Memory Model Feature Register 1, EL1* on page B2-391.
- *B2.74 ID_MMFR3_EL1, AArch32 Memory Model Feature Register 3, EL1* on page B2-395.
- *B2.75 ID_MMFR4_EL1, AArch32 Memory Model Feature Register 4, EL1* on page B2-397.

Bit fields and details that are not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

B2.74 ID_MMFR3_EL1, AArch32 Memory Model Feature Register 3, EL1

The ID_MMFR3_EL1 provides information about the memory model and memory management support in AArch32.

Bit field descriptions

ID_MMFR3_EL1 is a 32-bit register, and is part of the Identification registers functional group.

This register is Read Only.

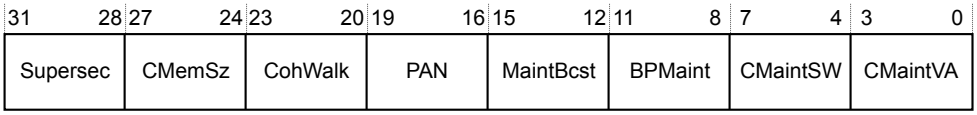

Figure B2-58 ID_MMFR3_EL1 bit assignments

Supersec, [31:28]
Supersections. Indicates support for supersections:

`0x0` Supersections supported.

CMemSz, [27:24]
Cached memory size. Indicates the size of physical memory supported by the core caches:

`0x2` 1TByte or more, corresponding to a 40-bit or larger physical address range.

CohWalk, [23:20]
Coherent walk. Indicates whether translation table updates require a clean to the point of unification:

`0x1` Updates to the translation tables do not require a clean to the point of unification to ensure visibility by subsequent translation table walks.

PAN, [19:16]
Privileged Access Never.

`0x2`
PAN supported and new `ATS1CPRP` and `ATS1CPWP` instructions supported.

MaintBcst, [15:12]
Maintenance broadcast. Indicates whether cache, TLB, and branch predictor operations are broadcast:

`0x2` Cache, TLB, and branch predictor operations affect structures according to shareability and defined behavior of instructions.

BPMaint, [11:8]
Branch predictor maintenance. Indicates the supported branch predictor maintenance operations.

`0x2` Supported branch predictor maintenance operations are:
- Invalidate all branch predictors.
- Invalidate branch predictors by MVA.

CMaintSW, [7:4]
Cache maintenance by set/way. Indicates the supported cache maintenance operations by set/way.

`0x1` Supported hierarchical cache maintenance operations by set/way are:

- Invalidate data cache by set/way.
- Clean data cache by set/way.
- Clean and invalidate data cache by set/way.

CMaintVA, [3:0]

Cache maintenance by *Virtual Address* (VA). Indicates the supported cache maintenance operations by VA.

`0x1` Supported hierarchical cache maintenance operations by VA are:

- Invalidate data cache by VA.
- Clean data cache by VA.
- Clean and invalidate data cache by VA.
- Invalidate instruction cache by VA.
- Invalidate all instruction cache entries.

Configurations

ID_MMFR3_EL1 is architecturally mapped to AArch32 register ID_MMFR3. See *B1.69 ID_MMFR3, Memory Model Feature Register 3* on page B1-233.

Must be interpreted with ID_MMFR0_EL1, ID_MMFR1_EL1, ID_MMFR2_EL1, and ID_MMFR4_EL1. See:

- *B2.71 ID_MMFR0_EL1, AArch32 Memory Model Feature Register 0, EL1* on page B2-389.
- *B2.72 ID_MMFR1_EL1, AArch32 Memory Model Feature Register 1, EL1* on page B2-391.
- *B2.73 ID_MMFR2_EL1, AArch32 Memory Model Feature Register 2, EL1* on page B2-393.
- *B2.75 ID_MMFR4_EL1, AArch32 Memory Model Feature Register 4, EL1* on page B2-397.

Bit fields and details that are not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

B2.75 ID_MMFR4_EL1, AArch32 Memory Model Feature Register 4, EL1

The ID_MMFR4_EL1 provides information about the memory model and memory management support in AArch32.

Bit field descriptions

ID_MMFR4_EL1 is a 32-bit register, and is part of the Identification registers functional group.

This register is Read Only.

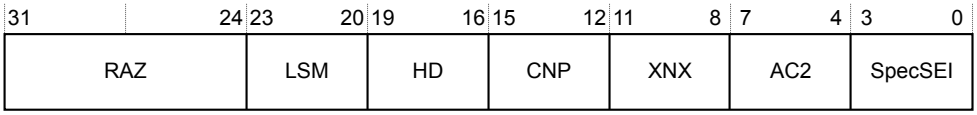

Figure B2-59 ID_MMFR4_EL1 bit assignments

RAZ, [31:24]

Read-As-Zero.

LSM, [23:20]

Load/Store Multiple. Indicates whether adjacent loads or stores can be combined. The value is:

`0x0` LSMAOE and nTLSMD bit not supported.

HD, [19:16]

Presence of Hierarchical Disables. Enables an operating system or hypervisor to hand over up to 4 bits of the last level page table descriptor (bits[62:59] of the page table entry) for use by hardware for IMPLEMENTATION DEFINED usage. The value is:

`0x2` Hierarchical Permission Disables and Hardware allocation of bits[62:59] supported.

CNP, [15:12]

Common Not Private. Indicates support for selective sharing of TLB entries across multiple PEs. The value is:

`0x1` CnP bit supported.

XNX, [11:8]

Execute Never. Indicates whether the stage 2 translation tables allows the stage 2 control of whether memory is executable at EL1 independent of whether memory is executable at EL0. The value is:

`0x1` EL0/EL1 execute control distinction at stage2 bit supported.

AC2, [7:4]

Indicates the extension of the ACTLR and HACTLR registers using ACTLR2 and HACTLR2. The value is:

`0x1` ACTLR2 and HACTLR2 are implemented.

SpecSEI, [3:0]

Describes whether the core can generate SError interrupt exceptions from speculative reads of memory, including speculative instruction fetches. The value is:

`0x0` The core never generates an SError interrupt due to an external abort on a speculative read.

Configurations

ID_MMFR4_EL1 is architecturally mapped to AArch64 register ID_MMFR4. See *B1.70 ID_MMFR4, Memory Model Feature Register 4* on page B1-235.

There is one copy of this register that is used in both Secure and Non-secure states.

Must be interpreted with ID_MMFR0_EL1, ID_MMFR1_EL1, ID_MMFR2_EL1, and ID_MMFR3_EL1. See:

- *B2.71 ID_MMFR0_EL1, AArch32 Memory Model Feature Register 0, EL1* on page B2-389.
- *B2.72 ID_MMFR1_EL1, AArch32 Memory Model Feature Register 1, EL1* on page B2-391.
- *B2.73 ID_MMFR2_EL1, AArch32 Memory Model Feature Register 2, EL1* on page B2-393.
- *B2.74 ID_MMFR3_EL1, AArch32 Memory Model Feature Register 3, EL1* on page B2-395.

Bit fields and details that are not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

B2.76 ID_PFR0_EL1, AArch32 Processor Feature Register 0, EL1

The ID_PFR0_EL1 provides top-level information about the instruction sets supported by the core in AArch32.

Bit field descriptions

ID_PFR0_EL1 is a 32-bit register, and is part of the Identification registers functional group.

This register is Read Only.

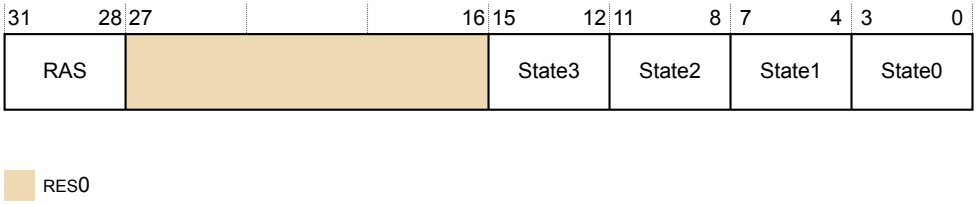

Figure B2-60 ID_PFR0_EL1 bit assignments

RAS, [31:28]
RAS extension version. The value is:

`0x1` Version 1 of the RAS extension is present.

RES0, [27:16]

RES0 Reserved.

State3, [15:12]
Indicates support for *Thumb Execution Environment* (T32EE) instruction set. This value is:

`0x0` Core does not support the T32EE instruction set.

State2, [11:8]
Indicates support for Jazelle. This value is:

`0x1` Core supports trivial implementation of Jazelle.

State1, [7:4]
Indicates support for T32 instruction set. This value is:

`0x3` Core supports T32 encoding after the introduction of Thumb-2 technology, and for all 16-bit and 32-bit T32 basic instructions.

State0, [3:0]
Indicates support for A32 instruction set. This value is:

`0x1` A32 instruction set implemented.

Configurations
ID_PFR0_EL1 is architecturally mapped to AArch32 register ID_PFR0. See *B1.71 ID_PFR0, Processor Feature Register 0* on page B1-237.

Bit fields and details that are not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

B2.77 ID_PFR1_EL1, AArch32 Core Feature Register 1, EL1

The ID_PFR1_EL1 provides information about the programmers model and architecture extensions supported by the core.

Bit field descriptions

ID_PFR1_EL1 is a 32-bit register, and is part of the Identification registers functional group.

This register is Read Only.

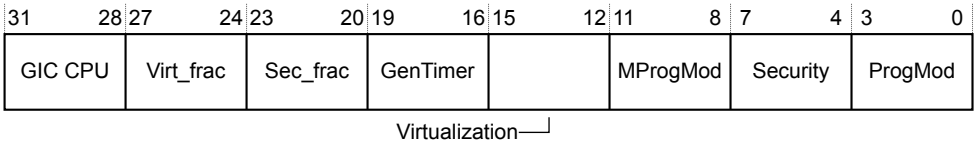

Figure B2-61 ID_PFR1_EL1 bit assignments

GIC CPU, [31:28]
GIC CPU support:

0 GIC CPU interface is disabled, **GICCDISABLE** is HIGH, or not implemented.

1 GIC CPU interface is implemented and enabled, **GICCDISABLE** is LOW.

Virt_frac, [27:24]

0 No features from the Armv7 Virtualization Extensions are implemented.

Sec_frac, [23:20]

0 No features from the Armv7 Virtualization Extensions are implemented.

GenTimer, [19:16]
Generic Timer support:

1 Generic Timer supported.

Virtualization, [15:12]
Virtualization support:

1 Virtualization implemented.

MProgMod, [11:8]
M profile programmers' model support:

0 Not supported.

Security, [7:4]
Security support:

1 Security implemented. This includes support for Monitor mode and the SMC instruction.

ProgMod, [3:0]
Indicates support for the standard programmers model for Armv4 and later.

Model must support User, FIQ, IRQ, Supervisor, Abort, Undefined, and System modes:

1 Supported.

Configurations
ID_PFR1_EL1 is architecturally mapped to AArch32 register ID_PFR1. See *B1.72 ID_PFR1, Processor Feature Register 1* on page B1-238.

B2.78 IFSR32_EL2, Instruction Fault Status Register, EL2

The IFSR32_EL2 allows access to the AArch32 IFSR register from AArch64 state only. Its value has no effect on execution in AArch64 state.

Bit field descriptions

IFSR32_EL2 is a 32-bit register, and is part of the Exception and fault handling registers functional group.

There are two formats for this register. The current translation table format determines which format of the register is used.

- *B2.78.1 IFSR32_EL2 with Short-descriptor translation table format* on page B2-401.
- *B2.78.2 IFSR32_EL2 with Long-descriptor translation table format* on page B2-401.

Configurations

IFSR32_EL2 is architecturally mapped to AArch32 register IFSR(NS). See *B1.73 IFSR, Instruction Fault Status Register* on page B1-240.

RW fields in this register reset to UNKNOWN values.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

This section contains the following subsections:

- *B2.78.1 IFSR32_EL2 with Short-descriptor translation table format* on page B2-401.
- *B2.78.2 IFSR32_EL2 with Long-descriptor translation table format* on page B2-401.

B2.78.1 IFSR32_EL2 with Short-descriptor translation table format

IFSR32_EL2 has a specific format when using the Short-descriptor translation table format.

The following figure shows the IFSR32_EL2 bit assignments when using the Short-descriptor translation table format.

When TTBCR.EAE==0:

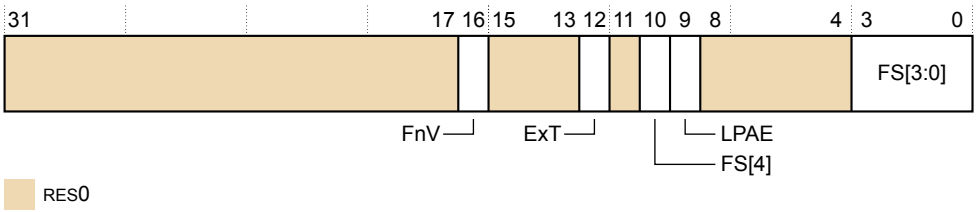

Figure B2-62 IFSR32_EL2 bit assignments for Short-descriptor translation table format

ExT, [12]

External abort type.

Read as zero.

For aborts other than external aborts, this bit always returns 0.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

B2.78.2 IFSR32_EL2 with Long-descriptor translation table format

IFSR32_EL2 has a specific format when using the Long-descriptor translation table format.

The following figure shows the IFSR32_EL2 bit assignments when using the Long-descriptor translation table format.

When TTBCR.EAE==1:

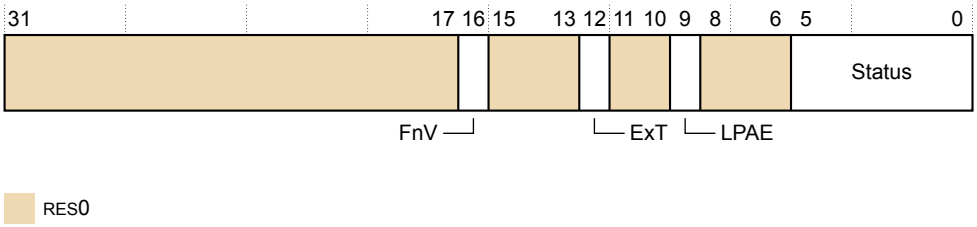

Figure B2-63 IFSR32_EL2 bit assignments for Long-descriptor translation table format

ExT, [12]

External abort type.

Read as zero.

For aborts other than external aborts, this bit always returns 0.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

B2.79 LORC_EL1, LORegion Control Register, EL1

The LORC_EL1 register enables and disables LORegions, and selects the current LORegion descriptor.

Bit field descriptions

LORC_EL1 is a 64-bit register and is part of the Virtual memory control registers functional group.

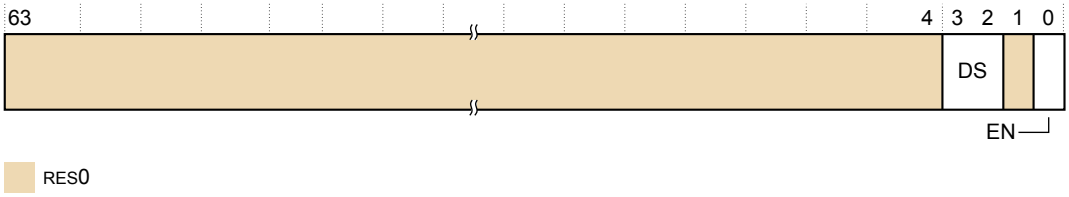

Figure B2-64 LORC_EL1 bit assignments

[63:4]
Reserved, RES0.

DS, [3:2]
Descriptor Select. Number that selects the current LORegion descriptor accessed by the LORSA_EL1, LOREA_EL1, and LORN_EL1 registers.

[1]
Reserved, RES0.

EN, [0]
Enable. The possible values are:

`0` Disabled. This is the reset value.

`1` Enabled.

Configurations

The LORC_EL1 register is only applicable to the AArch64 state and is not accessible at any Exception level in AArch32.

RW fields in this register reset to architecturally UNKNOWN values.

Bit fields and details that are not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

B2.80 LORID_EL1, LORegion ID Register, EL1

The LORID_EL1 ID register indicates the supported number of LORegions and LORegion descriptors.

Bit field descriptions

LORID_EL1 is a 64-bit register.

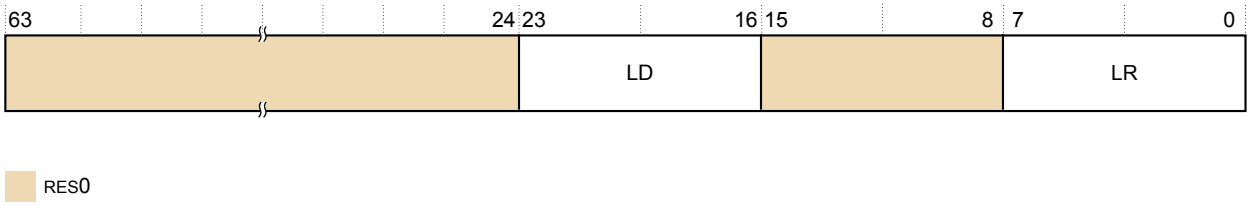

Figure B2-65 LORID_EL1 bit assignments

[63:24]
Reserved, RES0.

LD, [23:16]
Number of LORegion descriptors supported by the implementation, expressed as binary 8-bit number. The value is:

`0x04`

[15:8]
Reserved, RES0.

LR, [7:0]
Number of LORegions supported by the implementation, expressed as a binary 8-bit number. The value is:

`0x04`

Configurations
Bit fields and details that are not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

B2.81 LORN_EL1, LORegion Number Register, EL1

The LORN_EL1 register holds the number of the LORegion described in the current LORegion descriptor selected by LORC_EL1.DS.

Bit field descriptions

LORN_EL1 is a 64-bit register and is part of the Virtual memory control registers functional group.

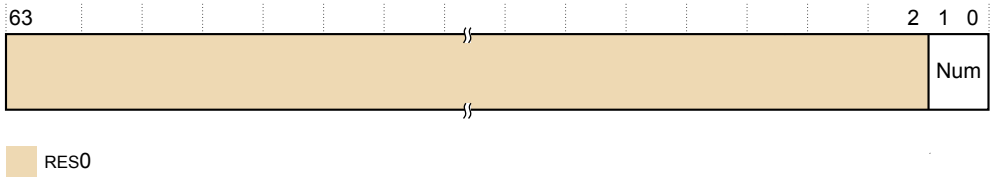

Figure B2-66 LORN_EL1 bit assignments

[63:2]

Reserved, RES0.

Num, [1:0]

Indicates the LORegion number.

Configurations

The LORN_EL1 register is only applicable to the AArch64 state and is not accessible at any Exception level in AArch32.

RW fields in this register reset to architecturally UNKNOWN values.

Bit fields and details that are not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

B2.82 MDCR_EL3, Monitor Debug Configuration Register, EL3

The MDCR_EL3 provides configuration options for Security to self-hosted debug.

Bit field descriptions

MDCR_EL3 is a 32-bit register, and is part of:

- The Debug registers functional group.
- The Security registers functional group.

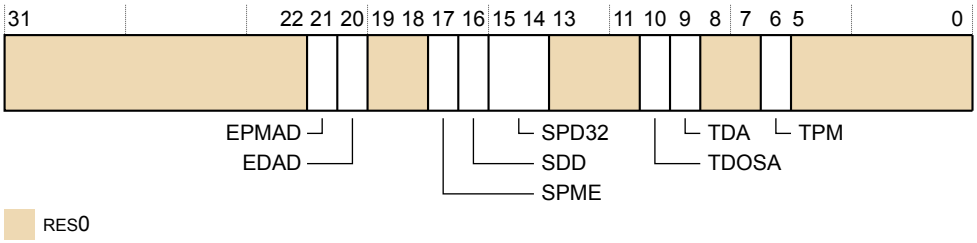

Figure B2-67 MDCR_EL3 bit assignments

EPMAD, [21]

External debugger access to Performance Monitors registers disabled. This disables access to these registers by an external debugger. The possible values are:

`0` Access to Performance Monitors registers from external debugger is permitted.

`1` Access to Performance Monitors registers from external debugger is disabled, unless overridden by authentication interface.

EDAD, [20]

External debugger access to breakpoint and watchpoint registers disabled. This disables access to these registers by an external debugger. The possible values are:

`0` Access to breakpoint and watchpoint registers from external debugger is permitted.

`1` Access to breakpoint and watchpoint registers from external debugger is disabled, unless overridden by authentication interface.

SPME, [17]

Secure performance monitors enable. This enables event counting exceptions from Secure state. The possible values are:

`0` Event counting prohibited in Secure state.

`1` Event counting allowed in Secure state.

SPD32, [15:14]

AArch32 secure privileged debug. Enables or disables debug exceptions from Secure state if Secure EL1 is using AArch32, other than software breakpoint instructions. The possible values are:

`0b00` Legacy mode. Debug exceptions from Secure EL1 are enabled only if `AArch32SelfHostedSecurePrivilegedInvasiveDebugEnabled()`.

`0b01` Reserved.

`0b10` Secure privileged debug disabled. Debug exceptions from Secure EL1 are disabled.

`0b11` Secure privileged debug enabled. Debug exceptions from Secure EL1 are enabled.

The reset value is UNKNOWN.

TDOSA, [10]

Trap accesses to the OS debug system registers, OSLAR_EL1, OSLSR_EL1, OSDLR_EL1, and DBGPRCR_EL1 OS.

`0` Accesses are not trapped.

`1` Accesses to the OS debug system registers are trapped to EL3.

The reset value is UNKNOWN.

TDA, [9]

Trap accesses to the remaining sets of debug registers to EL3.

`0` Accesses are not trapped.

`1` Accesses to the remaining debug system registers are trapped to EL3.

The reset value is UNKNOWN.

Bit fields and details that are not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

B2.83 MIDR_EL1, Main ID Register, EL1

The MIDR_EL1 provides identification information for the core, including an implementer code for the device and a device ID number.

Bit field descriptions

MIDR_EL1 is a 32-bit register, and is part of the Identification registers functional group.

This register is Read Only.

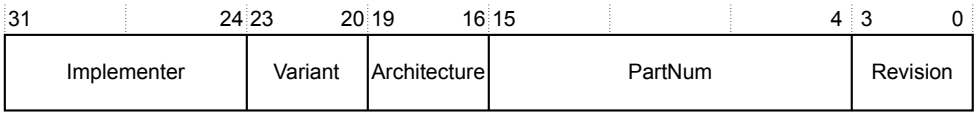

Figure B2-68 MIDR_EL1 bit assignments

Implementer, [31:24]
Indicates the implementer code. This value is:

`0x41` ASCII character 'A' - implementer is Arm Limited.

Variant, [23:20]
Indicates the variant number of the core. This is the major revision number *x* in the r*x* part of the r*x*p*y* description of the product revision status. This value is:

`0x2` r2p1.

Architecture, [19:16]
Indicates the architecture code. This value is:

`0xF` Defined by CPUID scheme.

PartNum, [15:4]
Indicates the primary part number. This value is:

`0xD0A` Cortex-A75 core.

Revision, [3:0]
Indicates the minor revision number of the core. This is the minor revision number *y* in the p*y* part of the r*x*p*y* description of the product revision status. This value is:

`0x1` r2p1.

Configurations
The MIDR_EL1 is:

- Architecturally mapped to the AArch32 MIDR register. See *B1.74 MIDR, Main ID Register* on page B1-242.
- Architecturally mapped to external MIDR_EL1 register.

Bit fields and details that are not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

B2.84 MPIDR_EL1, Multiprocessor Affinity Register, EL1

The MPIDR_EL1 provides an additional core identification mechanism for scheduling purposes in a cluster.

Bit field descriptions

MPIDR_EL1 is a 64-bit register, and is part of the Other system control registers functional group.

This register is Read Only.

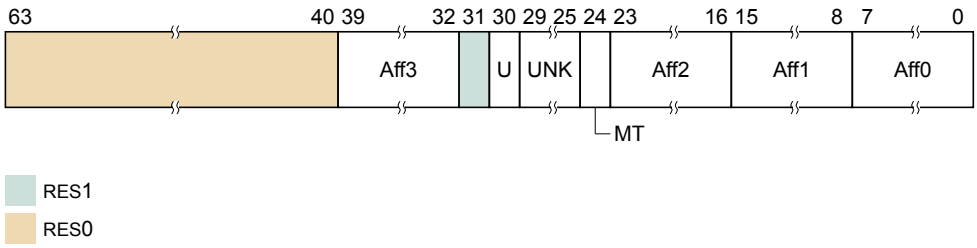

Figure B2-69 MPIDR_EL1 bit assignments

RES0, [63:40]

RES0 Reserved.

Aff3, [39:32]

Affinity level 3. Highest level affinity field.

CLUSTERID

Indicates the value read in the **CLUSTERIDAFF3** configuration signal.

RES1, [31]

RES1 Reserved.

U, [30]

Indicates a single core system, as distinct from core 0 in a cluster. This value is:

`0` Core is part of a multiprocessor system. This is the value for implementations with more than one core, and for implementations with an ACE or CHI master interface.

[29:25]

UNK Reserved.

MT, [24]

Indicates whether the lowest level of affinity consists of logical cores that are implemented using a multi-threading type approach. This value is:

`1` Performance of PEs at the lowest affinity level is very interdependent.

Affinity0 represents threads. Cortex-A75 is not multithreaded, but may be in a system with other cores that are multithreaded.

Aff2, [23:16]

Affinity level 2. Second highest level affinity field.

CLUSTERID

Indicates the value read in the **CLUSTERIDAFF2** configuration signal.

Aff1, [15:12]

Part of Affinity level 1. Third highest level affinity field.

RAZ Read-As-Zero.

Aff1, [11:8]

Part of Affinity level 1. Third highest level affinity field.

CPUID. Identification number for each CPU in the Cortex-A75 cluster:

`0x0` MP1: CPUID: 0. to

`0x7` MP8: CPUID: 7.

Aff0, [7:0]

Affinity level 0. The level identifies individual threads within a multi-threaded core. The Cortex-A75 core is single-threaded, so this field has the value `0x00`.

Configurations

MPIDR_EL1[31:0] is:

- Architecturally mapped to AArch32 register MPIDR. See *B1.75 MPIDR, Multiprocessor Affinity Register* on page B1-243.
- Mapped to external register EDDEVAFF0.

MPIDR_EL1[63:32] is mapped to external register EDDEVAFF1.

Bit fields and details that are not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

B2.85 PAR_EL1, Physical Address Register, EL1

The PAR_EL1 returns the output address from an address translation instruction that executed successfully, or fault information if the instruction did not execute successfully.

Bit field descriptions, PAR_EL1.F is 0

The following figure shows the PAR bit assignments when PAR.F is 0.

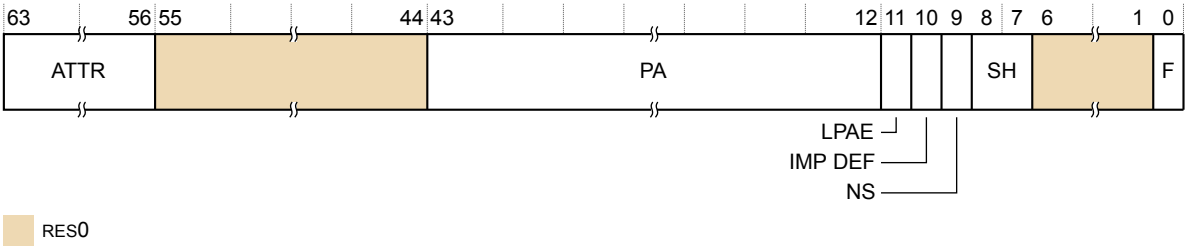

Figure B2-70 PAR bit assignments, PAR_EL1.F is 0

IMP DEF, [10]

IMPLEMENTATION DEFINED. Bit[10] is RES0.

F, [0]

Indicates whether the instruction performed a successful address translation.

`0` Address translation completed successfully.

`1` Address translation aborted.

Bit field descriptions, PAR_EL1.F is 1

See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

B2.86 REVIDR_EL1, Revision ID Register, EL1

The REVIDR_EL1 provides revision information, additional to MIDR_EL1, that identifies minor fixes (errata) which might be present in a specific implementation of the Cortex-A75 core.

Bit field descriptions

REVIDR_EL1 is a 32-bit register, and is part of the Identification registers functional group.

This register resets to value `0x00000000`.

This register is Read Only.

Figure B2-71 REVIDR_EL1 bit assignments

IMPLEMENTATION DEFINED, [31:0]

IMPLEMENTATION DEFINED.

Configurations

REVIDR_EL1 is architecturally mapped to AArch32 register REVIDR. See *B1.79 REVIDR, Revision ID Register* on page B1-249.

Bit fields and details that are not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

B2.87 RMR_EL3, Reset Management Register

The RMR_EL3 controls the execution state that the core boots into and allows request of a Warm reset.

When this register is implemented:

- A write to the register can request a Warm reset.
- If EL3 can use AArch32 and AArch64, this register specifies the Execution state that the PE boots into on a Warm reset.

Bit field descriptions

RMR_EL1 is a 32-bit register, and is part of the Reset management registers functional group.

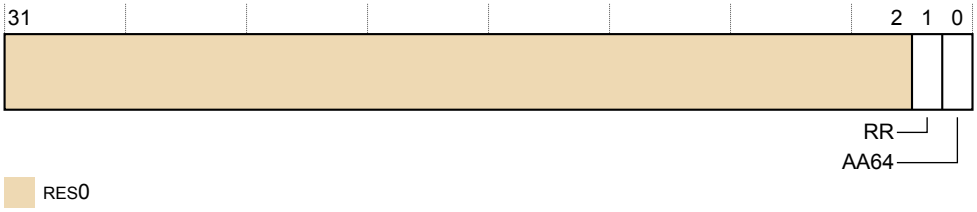

Figure B2-72 RMR_EL3 bit assignments

RES0, [31:2]

RES0 Reserved.

RR, [1]

Reset Request. The possible values are:

0 This is the reset value on both a Warm and a Cold reset.

1 Requests a Warm reset.

The bit is strictly a request.

AA64, [0]

Determines which execution state the core boots into after a Warm reset. The possible values are:

0 AArch32 Execution state.

1 AArch64 Execution state.

The reset vector address on reset takes a choice between two values, depending on the value in the AA64 bit. This ensures that even with reprogramming of the AA64 bit, it is not possible to change the reset vector to go to a different location.

The Cold reset value depends on the **AA64nAA32** configuration signal.

On coming out of the Warm reset, execution starts at the IMPLEMENTATION DEFINED reset vector address of the specified Execution state.

If EL3 cannot use AArch32 this bit is RAO/WI.

When implemented as RW, this field resets to 1 on a Cold reset.

Configurations

The RMR_EL3 is architecturally mapped to the AArch32 RMR register.

When EL3 is implemented:

- If EL3 can use AArch32 and AArch64, then this register must be implemented.
- If EL3 cannot use AArch32 then it is IMPLEMENTATION DEFINED whether the register is implemented.

When this register is not implemented its encoding is UNDEFINED.

Details that are not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

B2.88 RVBAR_EL3, Reset Vector Base Address Register, EL3

If EL3 is the highest Exception level implemented, RVBAR_EL3 contains the IMPLEMENTATION DEFINED address that execution starts from after reset when executing in AArch64 state.

Bit field descriptions

RVBAR_EL3 is a 64-bit register, and is part of the Reset management registers functional group.

This register is Read Only.

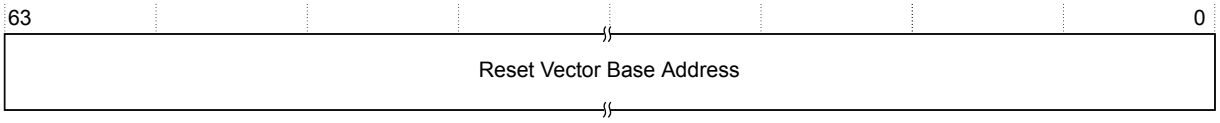

Figure B2-73 RVBAR_EL3 bit assignments

RVBA, [63:0]

Reset Vector Base Address. The address that execution starts from after reset when executing in 64-bit state. Bits[1:0] of this register are `0b00`, as this address must be aligned, and bits [63:40] are `0x000000` because the address must be within the physical address size supported by the core.

Configurations

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

B2.89 SCTLR_EL1, System Control Register, EL1

The SCTLR_EL1 provides top-level control of the system, including its memory system, at EL1 and EL0.

Bit field descriptions

SCTLR_EL1 is a 32-bit register, and is part of the Other system control registers functional group.

Apart from bits[25], [12], [2], and [0], this register has UNKNOWN reset values.

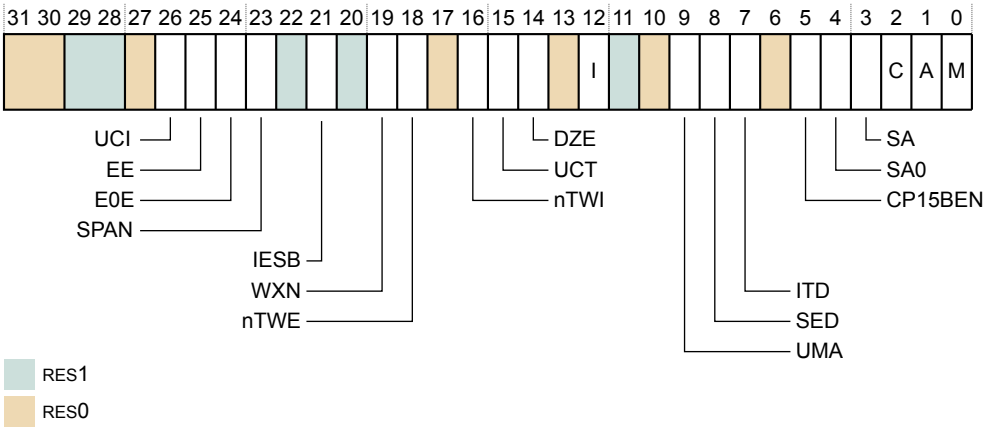

Figure B2-74 SCTLR_EL1 bit assignments

RES0, [31:30]

RES0 Reserved.

RES1, [29:28]

RES1 Reserved.

RES0, [27]

RES0 Reserved.

EE, [25]

Exception endianness. The value of this bit controls the endianness for explicit data accesses at EL1. This value also indicates the endianness of the translation table data for translation table lookups. The possible values of this bit are:

`0` Little-endian.

`1` Big-endian.

The reset value of this bit is determined by the CFGEND configuration pin.

ITD, [7]

This field is RAZ/WI.

RES0, [6]

RES0 Reserved.

CP15BEN, [5]

CP15 barrier enable. The possible values are:

`0` CP15 barrier operations disabled. Their encodings are UNDEFINED.

`1` CP15 barrier operations enabled.

M, [0]

MMU enable. The possible values are:

`0` EL1 and EL0 stage 1 MMU disabled.

`1` EL1 and EL0 stage 1 MMU enabled.

Configurations

SCTLR_EL1 is architecturally mapped to AArch32 register SCTLR(NS) See *B1.82 SCTLR, System Control Register* on page B1-252.

Bit fields and details that are not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

B2.90 SCTLR_EL2, System Control Register, EL2

The SCTLR_EL2 provides top-level control of the system, including its memory system at EL2.

Bit field descriptions

SCTLR_EL2 is a 32-bit register, and is part of:

- The Virtualization registers functional group.
- The Other system control registers functional group.

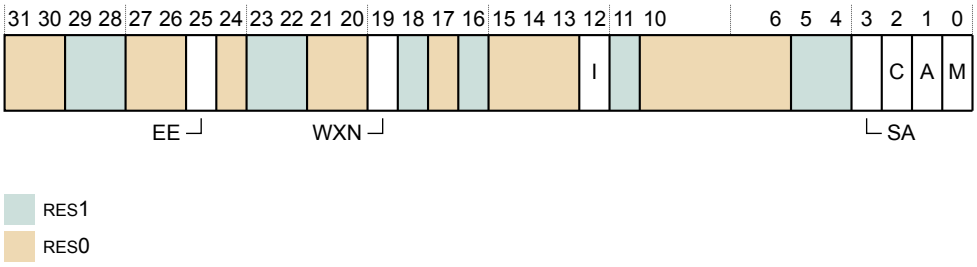

Figure B2-75 SCTLR_EL2 bit assignments

Apart from bits [12], [2], and [0], this register resets to UNKNOWN values.

Configurations

SCTLR_EL2 is architecturally mapped to AArch32 register HSCTLR. See *B1.54 HSCTLR, Hyp System Control Register* on page B1-206.

If EL2 is not implemented, this register is RES0 from EL3.

Bit fields and details that are not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

B2.91 SCTLR_EL3, System Control Register, EL3

The SCTLR_EL3 provides top-level control of the system, including its memory system at EL3.

Bit field descriptions

SCTLR_EL3 is a 32-bit register, and is part of the Other system control registers functional group.

Apart from bits [25] ,[12], [2], and [0], this register has UNKNOWN reset values.

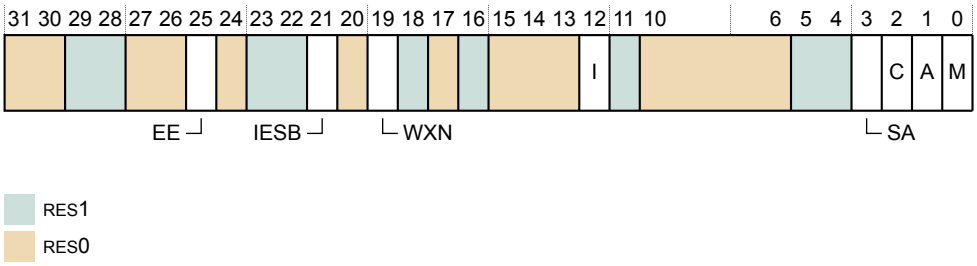

Figure B2-76 SCTLR_EL3 bit assignments

RES0, [31:30]

RES0 Reserved.

RES1, [29:28]

RES1 Reserved.

RES0, [27:26]

RES0 Reserved.

EE, [25]

Exception endianness. This bit controls the endianness for:

- Explicit data accesses at EL3.
- Stage 1 translation table walks at EL3.

The possible values are:

0 Little endian.

1 Big endian.

The reset value is determined by the CFGEND configuration signal.

I, [12]

Global instruction cache enable. The possible values are:

0 Instruction caches disabled. This is the reset value.

1 Instruction caches enabled.

C, [2]

Global enable for data and unifies caches. The possible values are:

0 Disables data and unified caches. This is the reset value.

1 Enables data and unified caches.

M, [0]

Global enable for the EL3 MMU. The possible values are:

0 Disables EL3 MMU. This is the reset value.

1 Enables EL3 MMU.

Configurations

Some or all RW fields of this register have defined reset values. These apply only if the PE resets into EL3 using AArch64. Otherwise, RW fields in this register reset to architecturally UNKNOWN values.

Bit fields and details that are not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

B2.92 TCR_EL1, Translation Control Register, EL1

The TCR_EL1 determines which Translation Base registers define the base address register for a translation table walk required for stage 1 translation of a memory access from EL0 or EL1 and holds cacheability and shareability information.

Bit field descriptions

TCR_EL1 is a 64-bit register, and is part of the Virtual memory control registers functional group.

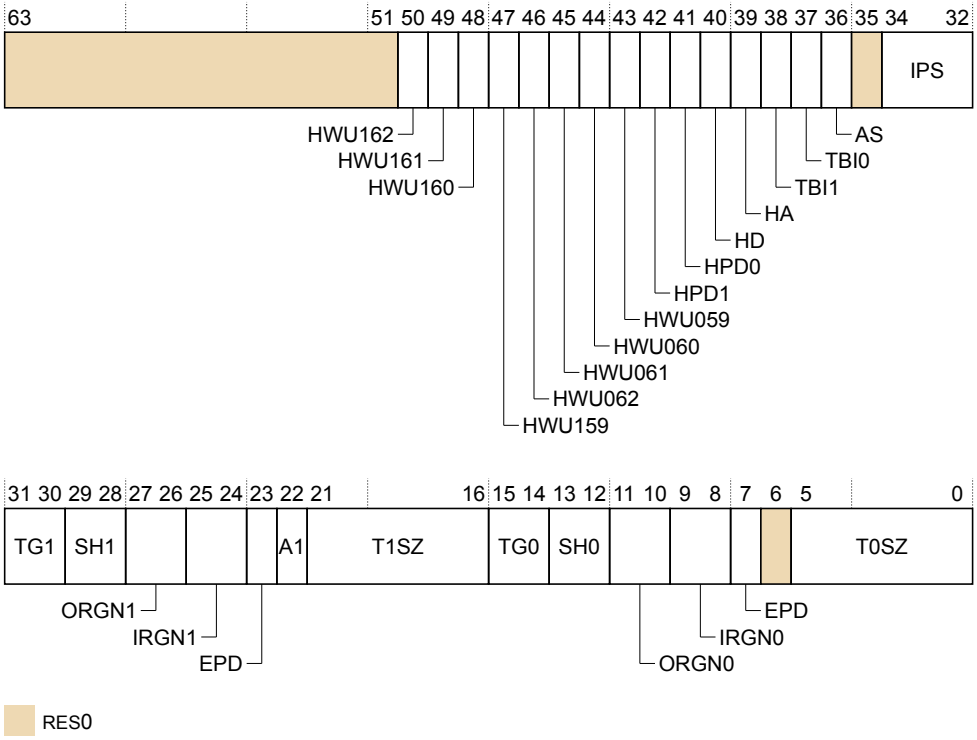

Figure B2-77 TCR_EL1 bit assignments

Note

Bits[50:39], architecturally defined, are implemented in the core.

HD, [40]

Hardware management of dirty state in stage 1 translations from EL0 and EL1. The possible values are:

`0` Stage 1 hardware management of dirty state disabled.

`1` Stage 1 hardware management of dirty state enabled, only if the HA bit is also set to 1.

HA, [39]

Hardware Access flag update in stage 1 translations from EL0 and EL1. The possible values are:

`0` Stage 1 Access flag update disabled.

`1` Stage 1 Access flag update enabled.

Configurations

TCR_EL1 is architecturally mapped to AArch32 register TTBCR(NS).

RW fields in this register reset to UNKNOWN values.

Bit fields and details that are not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

B2.93 TCR_EL2, Translation Control Register, EL2

The TCR_EL2 controls translation table walks required for stage 1 translation of a memory access from EL2 and holds cacheability and shareability information.

Bit field descriptions

TCR_EL2 is a 32-bit register.

TCR_EL2 is part of:

- The Virtual memory control registers functional group.
- The Hypervisor and virtualization registers functional group.

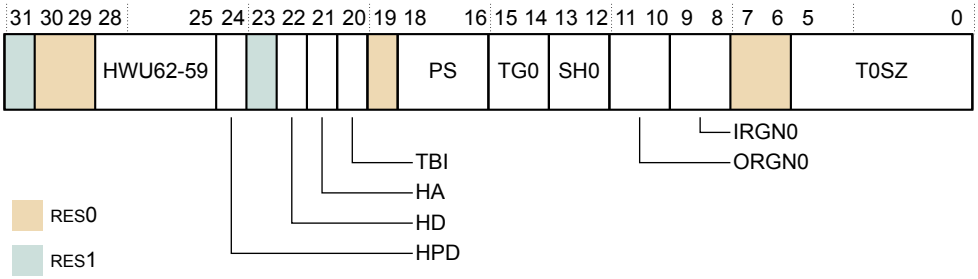

Figure B2-78 TCR_EL2 bit assignments

Note

Bits[28:21], architecturally defined, are implemented in the core.

HD, [22]

Dirty bit update. The possible values are:

`0` Dirty bit update is disabled.

`1` Dirty bit update is enabled.

HA, [21]

Stage 1 Access flag update. The possible values are:

`0` Stage 1 Access flag update is enabled.

`1` Stage 1 Access flag update is disabled.

Configurations

TCR_EL2 is architecturally mapped to AArch32 register HCTR.

When the Virtualization Host Extension is activated, TCR_EL2 has the same bit assignments as TCR_EL1.

Bit fields and details that are not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

B2.94 TCR_EL3, Translation Control Register, EL3

The TCR_EL3 controls translation table walks required for stage 1 translation of memory accesses from EL3 and holds cacheability and shareability information for the accesses.

Bit field descriptions

TCR_EL3 is a 32-bit register and is part of the Virtual memory control registers functional group.

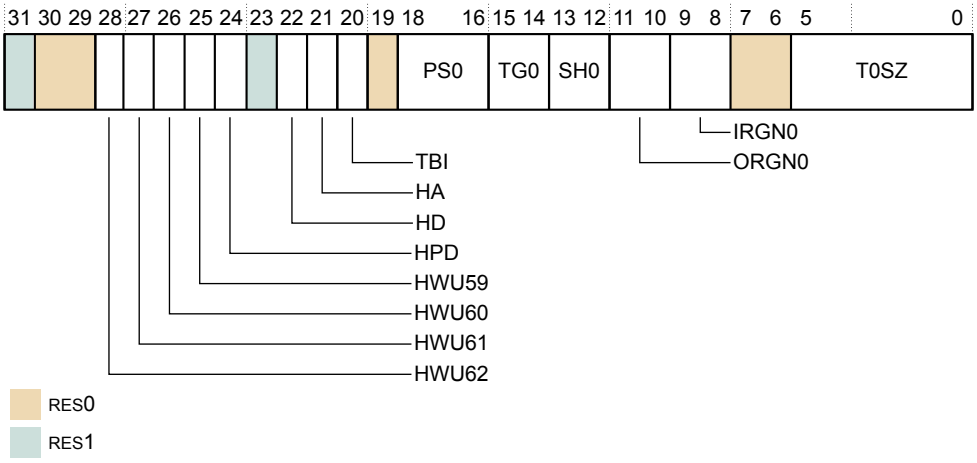

Figure B2-79 TCR_EL3 bit assignments

Note

Bits[28:21], architecturally defined, are implemented in the core.

HD, [22]

Dirty bit update. The possible values are:

`0` Dirty bit update is disabled.

`1` Dirty bit update is enabled.

HA, [21]

Stage 1 Access flag update. The possible values are:

`0` Stage 1 Access flag update is enabled.

`1` Stage 1 Access flag update is disabled.

Configurations

TCR_EL3 is mapped to AArch32 register TTBR(S). See *B1.85 TTBR0, Translation Table Base Register 0* on page B1-258.

Bit fields and details that are not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

B2.95 TTBR0_EL1, Translation Table Base Register 0, EL1

The TTBR0_EL1 holds the base address of translation table 0, and information about the memory it occupies. This is one of the translation tables for the stage 1 translation of memory accesses from modes other than Hyp mode.

Bit field descriptions

TTBR0_EL1 is 64-bit register.

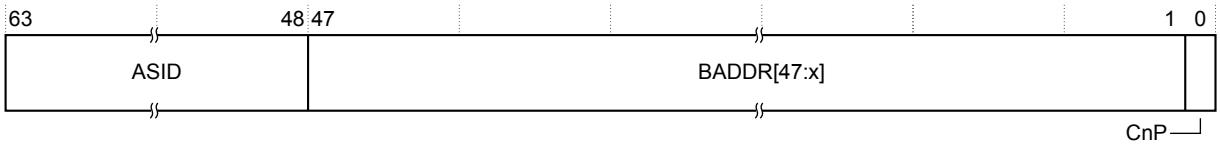

Figure B2-80 TTBR0_EL1 bit assignments

ASID, [63:48]
An ASID for the translation table base address. The TCR_EL1.A1 field selects either TTBR0_EL1.ASID or TTBR1_EL1.ASID.

BADDR[47:x], [47:1]
Translation table base address, bits[47:x]. Bits [x-1:1] are RES0.

x is based on the value of TCR_EL1.T0SZ, the stage of translation, and the memory translation granule size.

For instructions on how to calculate it, see the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The value of x determines the required alignment of the translation table, that must be aligned to 2^x bytes.

If bits [x-1:1] are not all zero, this is a misaligned translation table base address. Its effects are CONSTRAINED UNPREDICTABLE, where bits [x-1:1] are treated as if all the bits are zero. The value read back from those bits is the value written.

CnP, [0]
Common not Private. The possible values are:

`0` CnP is not supported.

`1` CnP is supported.

Configurations
There are no configuration notes.
Bit fields and details that are not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

B2.96 TTBR0_EL2, Translation Table Base Register 0, EL2

The TTBR0_EL2 holds the base address of the translation table for the stage 1 translation of memory accesses from EL2.

Bit field descriptions

TTBR0_EL2 is a 64-bit register, and is part of the Virtual memory control registers functional group.

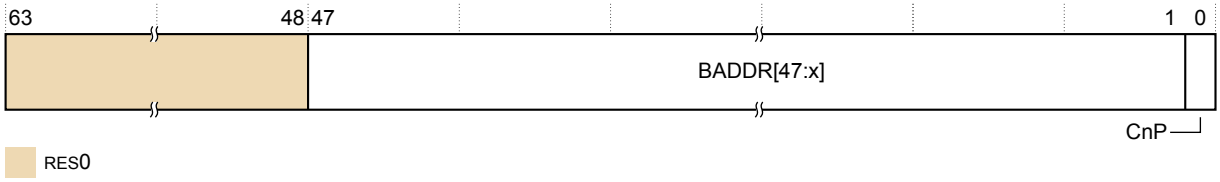

Figure B2-81 TTBR0_EL2 bit assignments

RES0, [63:48]

RES0 Reserved.

BADDR, [47:1]

Translation table base address, bits[47:x]. Bits [x-1:1] are RES0.

x is based on the value of TCR_EL2.T0SZ, the stage of translation, and the memory translation granule size.

For instructions on how to calculate it, see the *Arm® Architecture Reference Manual Arm®v8, for Arm®v8-A architecture profile*.

The value of x determines the required alignment of the translation table, that must be aligned to 2^x bytes.

If bits [x-1:1] are not all zero, this is a misaligned translation table base address. Its effects are CONSTRAINED UNPREDICTABLE, where bits [x-1:1] are treated as if all the bits are zero. The value read back from those bits is the value written.

CnP, [0]

Common not Private. The possible values are:

`0` CnP is not supported.

`1` CnP is supported.

Configurations

TTBR0_EL2 is architecturally mapped to AArch32 register HTTBR, Hyp Translation Table Base Register.

When the Virtualization Host Extension is activated, TTBR0_EL2 has the same bit assignments as TTBR0_EL1.

Bit fields and details that are not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

B2.97 TTBR0_EL3, Translation Table Base Register 0, EL3

The TTBR0_EL3 holds the base address of the translation table for the stage 1 translation of memory accesses from EL3.

Bit field descriptions

TTBR0_EL3 is a 64-bit register.

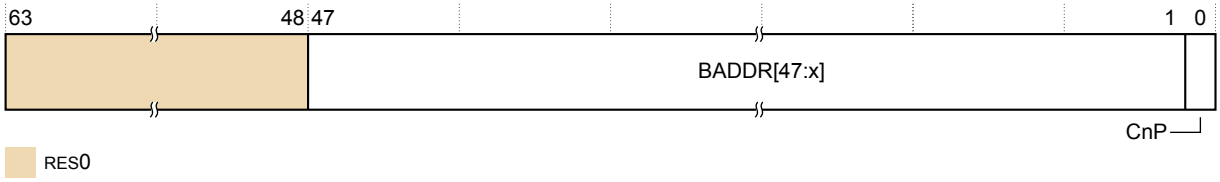

Figure B2-82 TTBR0_EL3 bit assignments

[63:48]
Reserved, RES0.

BADDR[47:x], [47:1]
Translation table base address, bits[47:x]. Bits [x-1:1] are RES0.

x is based on the value of TCR_EL1.T0SZ, the stage of translation, and the memory translation granule size.

For instructions on how to calculate it, see the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The value of x determines the required alignment of the translation table, that must be aligned to 2^x bytes.

If bits [x-1:1] are not all zero, this is a misaligned translation table base address. Its effects are CONSTRAINED UNPREDICTABLE, where bits [x-1:1] are treated as if all the bits are zero. The value read back from those bits is the value written.

CnP, [0]
Common not Private. The possible values are:

`0` CnP is not supported.

`1` CnP is supported.

Configurations
TTBR0_EL3 is mapped to AArch32 register TTBR0 (S). See *B1.85 TTBR0, Translation Table Base Register 0* on page B1-258.

Bit fields and details that are not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

B2.98 TTBR1_EL1, Translation Table Base Register 1, EL1

The TTBR1_EL1 holds the base address of translation table 1, and information about the memory it occupies. This is one of the translation tables for the stage 1 translation of memory accesses at EL0 and EL1.

Bit field descriptions

TTBR1_EL1 is a 64-bit register.

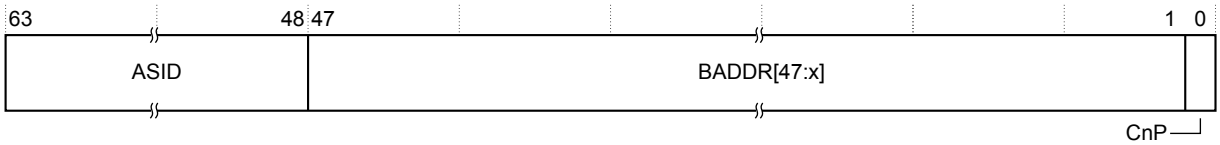

Figure B2-83 TTBR1_EL1 bit assignments

ASID, [63:48]
An ASID for the translation table base address. The TCR_EL1.A1 field selects either TTBR0_EL1.ASID or TTBR1_EL1.ASID.

BADDR[47:x], [47:1]
Translation table base address, bits[47:x]. Bits [x-1:0] are RES0.

x is based on the value of TCR_EL1.T0SZ, the stage of translation, and the memory translation granule size.

For instructions on how to calculate it, see the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The value of x determines the required alignment of the translation table, that must be aligned to 2^x bytes.

If bits [x-1:1] are not all zero, this is a misaligned Translation Table Base Address. Its effects are CONSTRAINED UNPREDICTABLE, where bits [x-1:1] are treated as if all the bits are zero. The value read back from those bits is the value written.

CnP, [0]
Common not Private. The possible values are:

`0` CnP is not supported.

`1` CnP is supported.

Configurations
TTBR1_EL1 is architecturally mapped to AArch32 register TTBR1 (NS). See *B1.86 TTBR1, Translation Table Base Register 1* on page B1-260.
Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

B2.99 TTBR1_EL2, Translation Table Base Register 1, EL2

TTBR1_EL2 has the same format and contents as TTBR1_EL1.

See *B2.98 TTBR1_EL1, Translation Table Base Register 1, EL1* on page B2-427.

B2.100 VDISR_EL2, Virtual Deferred Interrupt Status Register, EL2

The VDISR_EL2 records that a virtual SError interrupt has been consumed by an ESB instruction executed at Non-secure EL1.

Bit field descriptions

VDISR_EL2 is a 64-bit register, and is part of the *Reliability, Availability, Serviceability* (RAS) registers functional group.

There are three formats for this register. The current translation table format determines which format of the register is used.

- When written at EL1 using short-descriptor format. See *B2.100.1 VDISR_EL2 with short-descriptor translation table format* on page B2-429.
- When written at EL1 using long-descriptor format. See *B2.100.2 VDISR_EL2 with long-descriptor translation table format* on page B2-430.
- When written at EL2. See *B2.100.3 VDISR_EL2 at EL1 using AArch64* on page B2-431.

Configurations

VDISR_EL2 is RES0 at EL3 if EL2 is not implemented.Present only if all the following are present and is UNDEFINED otherwise.

If the implementation supports AArch32 at EL2, VDISR_EL2 is architecturally mapped to VDISR. See *B1.89 VDISR, Virtual Deferred Interrupt Status Register* on page B1-264.

Bit fields and details that are not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

This section contains the following subsections:

- *B2.100.1 VDISR_EL2 with short-descriptor translation table format* on page B2-429.
- *B2.100.2 VDISR_EL2 with long-descriptor translation table format* on page B2-430.
- *B2.100.3 VDISR_EL2 at EL1 using AArch64* on page B2-431.

B2.100.1 VDISR_EL2 with short-descriptor translation table format

VDISR_EL2 has a specific format when using the Short-descriptor translation table format.

Bit field descriptions

The following figure shows the VDISR_EL2 bit assignments when EL1 is using the AArch32 Short-descriptor translation table format.

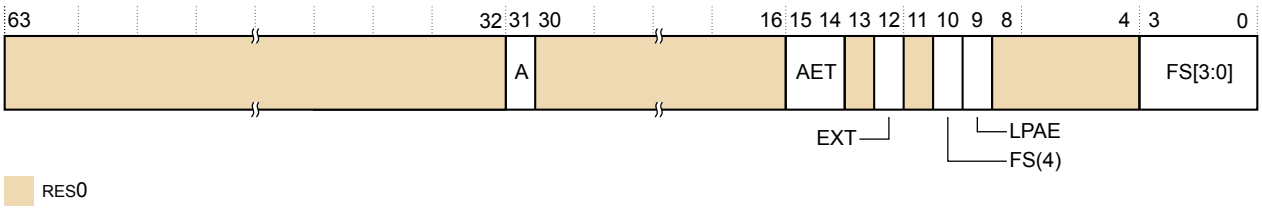

Figure B2-84 VDISR_EL2 bit assignments for Short-descriptor translation table format

RES0, [63:32]

RES0 Reserved.

A, [31]

Set to 1 when ESB defers a virtual SError interrupt.

RES0, [30:16]

RES0 Reserved.

AET, [15:14]
Asynchronous Error Type. Describes the state of the PE after taking an asynchronous Data Abort exception. The value is:

`0b00` *Uncorrected error, Uncontainable* (UC).

RES0, [13]

RES0 Reserved.

EXT, [12]
External Abort Type. This bit is defined as RES0.

RES0, [11]

RES0 Reserved.

FS(4), [10,3:0]
Fault status code. Set to `0b10110` when ESB defers a virtual SError interrupt. The value of this field is:

`0b10110` Asynchronous SError interrupt.

LPAE, [9]
Format. The value is:

`0b0` Using the Short-descriptor translation table format.

RES0, [8:4]

RES0 Reserved.

B2.100.2 VDISR_EL2 with long-descriptor translation table format

VDISR_EL2 has a specific format when using the Long-descriptor translation table format.

Bit field descriptions

The following figure shows the VDISR_EL2 bit assignments when EL1 is using the AArch32 Long-descriptor format.

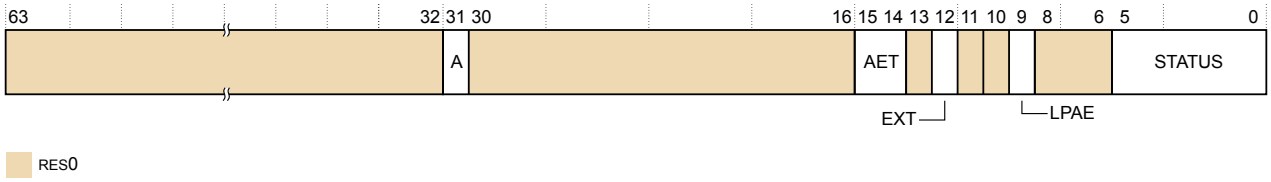

Figure B2-85 VDISR_EL2 bit assignments for the Long-descriptor format

RES0, [63:32]

RES0 Reserved.

A, [31]
Set to 1 when ESB defers a virtual SError interrupt.

RES0, [30:16]

RES0 Reserved.

AET, [15:14]
Contains the value from VDFSR.AET.

RES0, [13]

RES0 Reserved.

EXT, [12]
Contains the value from VDFSR.ExT.

RES0, [11]

RES0 Reserved.

RES0, [10]

RES0 Reserved.

LPAE, [9]
Format. The value is:

1 Using the Long-descriptor translation table format.

RES0, [8:6]

RES0 Reserved.

STATUS, [5:0]
Fault status code. Set to `0b010001` when ESB defers a virtual SError interrupt. The value of this field is:

`0b010001` Asynchronous SError interrupt.

B2.100.3 VDISR_EL2 at EL1 using AArch64

VDISR_EL2 has a specific format when written at EL1.

The following figure shows the VDISR_EL2 bit assignments when written at EL1 using AArch64:

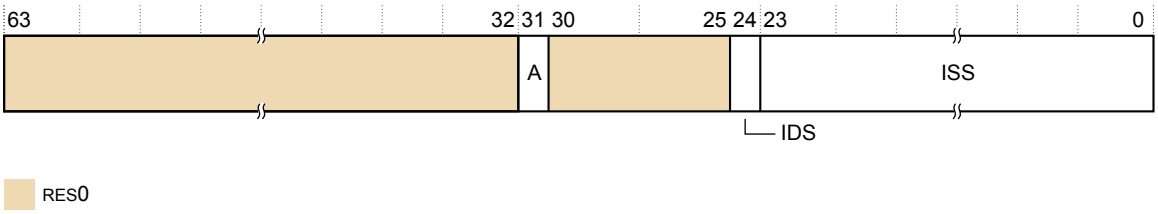

Figure B2-86 VDISR_EL2 at EL1 using AArch64

RES0, [63:32]

RES0 Reserved.

A, [31]
Set to 1 when ESB defers an asynchronous SError interrupt.

RES0, [30:25]

RES0 Reserved.

IDS, [24]
Contains the value from VSESR_EL2.IDS.

ISS, [23:0]
Contains the value from VSESR_EL2, bits[23:0].

B2.101 VSESR_EL2, Virtual SError Exception Syndrome Register

The VSESR_EL2 provides the syndrome value reported to software on taking a virtual SError interrupt exception.

Bit field descriptions

VSESR_EL2 is a 64-bit register, and is part of :

- The Exception and fault handling registers functional group.
- The Virtualization registers functional group.

The register has two bit assignment configurations, that depend on whether the virtual SError interrupt is taken to EL1 using AArch32 or AArch64:

- If the virtual SError interrupt is taken to EL1 using AArch64, VSESR_EL2 provides the syndrome value reported in ESR_EL1.
- If the virtual SError interrupt is taken to EL1 using AArch32, VSESR_EL2 provides the syndrome values reported in DFSR bits

VSESR_EL2 bit assignments when EL1 is using AArch32

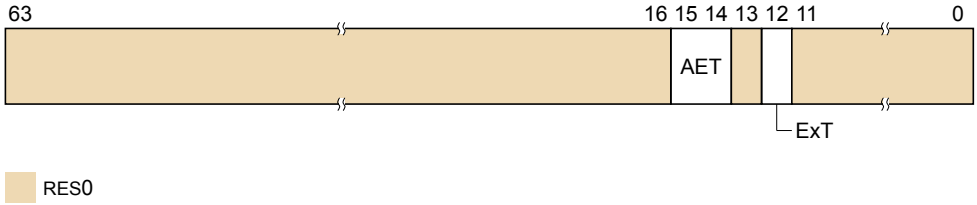

Figure B2-87 VSESR_EL2 bit assignments when EL1 is using AArch32

RES0, [63:16]

RES0 Reserved.

AET, [15:14]

Asynchronous Error Type. Describes the state of the core after taking the SError interrupt exception. Software might use the information in the syndrome registers to determine what recovery might be possible.

RES0, [13]

RES0 Reserved.

ExT, [12]

External abort type.

RES0, [11:0]

RES0 Reserved.

VSESR_EL2 bit assignments when EL1 is using AArch64

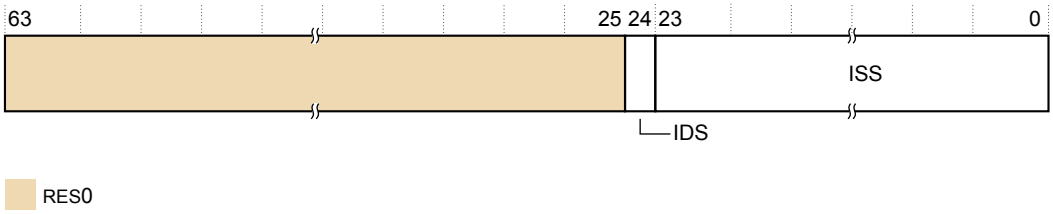

Figure B2-88 VSESR_EL2 bit assignments when EL1 is using AArch64

RES0, [63:25]

RES0 Reserved.

IDS, [24]

Indicates whether the deferred SError interrupt was of an IMPLEMENTATION DEFINED type. See ESR_EL1.IDS for a description of the functionality.

On taking a virtual SError interrupt to EL1 using AArch64 due to HCR_EL2.VSE == 1, ESR_EL1[24] is set to VSESR_EL2.IDS.

ISS, [23:0]

Syndrome information.See ESR_EL1.ISS for a description of the functionality.

On taking a virtual SError interrupt to EL1 using AArch32 due to HCR_EL2.VSE == 1, ESR_EL1 [23:0] is set to VSESR_EL2.ISS.

Configurations

AArch64 System register VSESR_EL2 [31:0] is architecturally mapped to AArch32 System register VDFSR. See *B1.88 VDFSR, Virtual SError Exception Syndrome Register on page B1-263*.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

B2.102 VTCR_EL2, Virtualization Translation Control Register, EL2

The VTCR_EL2 controls the translation table walks required for the stage 2 translation of memory accesses from Non-secure EL0 and EL1.

It also holds cacheability and shareability information for the accesses.

Bit field descriptions

VTCR_EL2 is a 32-bit register, and is part of:

- The Virtualization registers functional group.
- The Virtual memory control registers functional group.

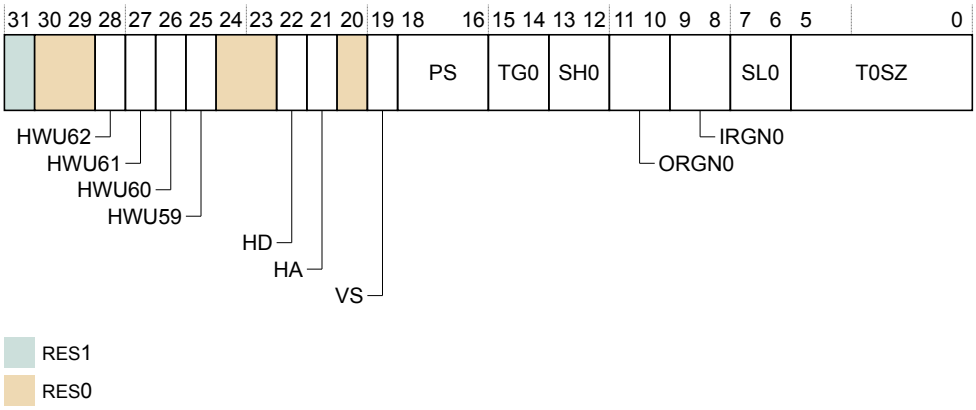

Figure B2-89 VTCR_EL2 bit assignments

Note

Bits[28:25] and bits[22:21], architecturally defined, are implemented in the core.

TG0, [15:14]

TTBR0_EL2 granule size. The possible values are:

`0b00` 4KB.

`0b01` 64KB.

`0b10` 16KB.

`0b11` Reserved.

All other values are not supported.

Configurations

VTCR_EL2 is architecturally mapped to AArch32 register VTCR. See *B1.92 VTCR, Virtualization Translation Control Register* on page B1-269.

RW fields in this register reset to architecturally UNKNOWN values.

Bit fields and details that are not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

B2.103 VTTBR_EL2, Virtualization Translation Table Base Register, EL2

VTTBR_EL2 holds the base address of the translation table for the stage 2 translation of memory accesses from Non-secure EL0 and EL1.

Bit field descriptions

VTTBR_EL2 is a 64-bit register.

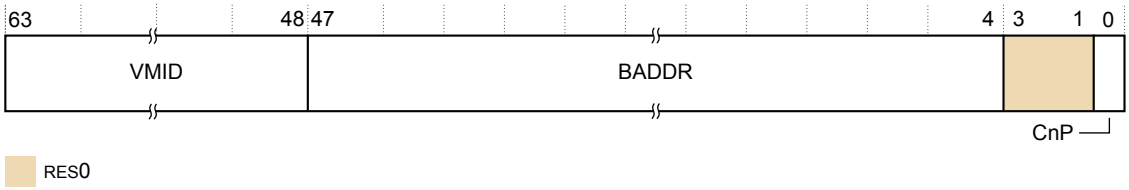

Figure B2-90 VTTBR_EL2 bit assignments

CnP, [0]

Common not Private. The possible values are:

`0` CnP is not supported.

`1` CnP is supported.

Configurations

There are no configuration notes.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

Chapter B3
Error system registers

This chapter describes the error registers accessed by both the AArch32 error registers and the AArch64 error registers.

It contains the following sections:

B3.1 Error system register summary

This section identifies the ERR0* core error record registers accessed by both the AArch32 and AArch64 ERX* error registers.

The ERR0* registers are agnostic to the architectural state. For example, this means that for ERRSELR==0 and ERRSELR_EL1==0, ERXPFGFR and ERXPFGFR_EL1 will both access ERR0PFGFR.

For those registers not described in this chapter, see the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The following table describes the architectural error record registers.

Table B3-1 Architectural error system register summary

Register mnemonic	Size	Register name	Access aliases from AArch32 and AArch64
ERR0ADDR	64	*B3.2 ERR0ADDR, Error Record Address Register* on page B3-440	*B1.30 ERXADDR, Selected Error Record Address Register* on page B1-178.
			B1.31 ERXADDR2, Selected Error Record Address Register 2 on page B1-179.
			B2.38 ERXADDR_EL1, Selected Error Record Address Register, EL1 on page B2-341
ERR0CTLR	64	*B3.3 ERR0CTLR, Error Record Control Register* on page B3-441	*B1.32 ERXCTLR, Selected Error Record Control Register* on page B1-180.
			B1.33 ERXCTLR2, Selected Error Record Control Register 2 on page B1-181.
			B2.39 ERXCTLR_EL1, Selected Error Record Control Register, EL1 on page B2-342
ERR0FR	64	*B3.4 ERR0FR, Error Record Feature Register* on page B3-443	*B1.34 ERXFR, Selected Error Record Feature Register* on page B1-182.
			B1.35 ERXFR2, Selected Error Record Feature Register 2 on page B1-183.
			B2.40 ERXFR_EL1, Selected Error Record Feature Register, EL1 on page B2-343
ERR0MISC0	64	*B3.5 ERR0MISC0, Error Record Miscellaneous Register 0* on page B3-445	*B1.36 ERXMISC0, Selected Error Miscellaneous Register 0* on page B1-184.
			B1.37 ERXMISC1, Selected Error Miscellaneous Register 1 on page B1-185.
			B2.41 ERXMISC0_EL1, Selected Error Record Miscellaneous Register 0, EL1 on page B2-344
ERR0MISC1	64	*B3.6 ERR0MISC1, Error Record Miscellaneous Register 1* on page B3-447	*B1.38 ERXMISC2, Selected Error Record Miscellaneous Register 2* on page B1-186 accesses bits [31:0]
			B1.39 ERXMISC3, Selected Error Record Miscellaneous Register 3 on page B1-187 accesses bits [63:32]
			B2.42 ERXMISC1_EL1, Selected Error Record Miscellaneous Register 1, EL1 on page B2-345

Table B3-1 Architectural error system register summary (continued)

Register mnemonic	Size	Register name	Access aliases from AArch32 and AArch64
ERR0STATUS	32	*B3.10 ERR0STATUS, Error Record Primary Status Register* on page B3-453	*B1.43 ERXSTATUS, Selected Error Record Primary Status Register* on page B1-193
			B2.46 ERXSTATUS_EL1, Selected Error Record Primary Status Register, EL1 on page B2-351

The following table describes the error record registers that are implementation defined.

Table B3-2 Implementation defined error system register summary

Register mnemonic	Size	Register name	Access aliases from AArch32 and AArch64
ERR0PFGCDNR	32	*B3.7 ERR0PFGCDNR, Error Pseudo Fault Generation Count Down Register* on page B3-448	*B1.40 ERXPFGCDNR, Selected Error Pseudo Fault Generation Count Down Register* on page B1-188
			B2.43 ERXPFGCDNR_EL1, Selected Error Pseudo Fault Generation Count Down Register, EL1 on page B2-346
ERR0PFGCTLR	32	*B3.8 ERR0PFGCTLR, Error Pseudo Fault Generation Control Register* on page B3-449	*B1.41 ERXPFGCTLR, Selected Error Pseudo Fault Generation Control Register* on page B1-190
			B2.44 ERXPFGCTLR_EL1, Selected Error Pseudo Fault Generation Control Register, EL1 on page B2-348
ERR0PFGFR	32	*B3.9 ERR0PFGFR, Error Pseudo Fault Generation Feature Register* on page B3-451	*B1.42 ERXPFGFR, Selected Pseudo Fault Generation Feature Register* on page B1-192
			B2.45 ERXPFGFR_EL1, Selected Pseudo Fault Generation Feature Register, EL1 on page B2-350

B3.2 ERR0ADDR, Error Record Address Register

The ERR0ADDR stores the address that is associated to an error that is recorded.

Bit field descriptions

ERR0ADDR is a 64-bit register, and is part of the *Reliability, Availability, Serviceability* (RAS) registers functional group.

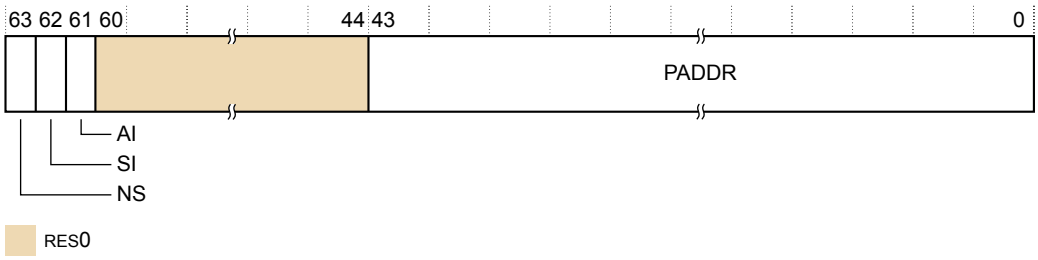

Figure B3-1 ERR0ADDR bit assignments

NS, [63]

Non-secure attribute. The possible values are:

`0` The physical address is Secure.

`1` The physical address is Non-secure.

SI, [62]

Secure Incorrect. Indicates if the NS bit is valid. The possible values are:

`0` The NS bit is correct. It matches the programmers view of the Non-secure attribute for this recorded location.

`1` The NS bit might not be correct, and might not match the programmers view of the Non-secure attribute for the recorded location.

AI, [61]

Address Incomplete or incorrect. Indicates whether the PADDR field is valid. The possible values are:

`0` The PADDR field is correct. It matches the view that the programmer has of the physical address for this recorded location.

`1` The PADDR field might not be correct, and might not match the view that the programmer has of the physical address for the recorded location.

RES0, [60:44]

RES0 Reserved.

PADDR, [43:0]

Physical address.

Configurations

ERR0ADDR resets to UNKNOWN.

This register is accessible from the following registers when ERRSELR.SEL==0:

- [31:0]: *B1.30 ERXADDR, Selected Error Record Address Register* on page B1-178.
- [63:32]: *B1.31 ERXADDR2, Selected Error Record Address Register 2* on page B1-179.
- *B2.38 ERXADDR_EL1, Selected Error Record Address Register, EL1* on page B2-341.

B3.3 ERR0CTLR, Error Record Control Register

The ERR0CTLR contains enable bits for the node that write to this record:

- Enabling error detection and correction.
- Enabling an error recovery interrupt.
- Enabling a fault handling interrupt.
- Enabling error recovery reporting as a read or write error response.

Bit field descriptions

ERR0CTLR is a 64-bit register and is part of the RAS registers functional group.

ERR0CTLR resets to `0b0000000000000000`.

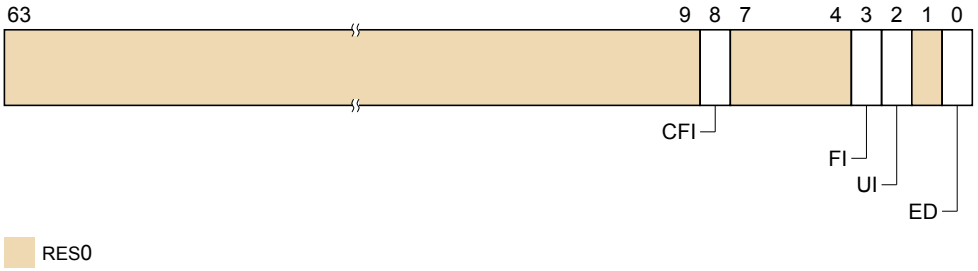

Figure B3-2 ERR0CTLR bit assignments

RES0, [63:9]

RES0 Reserved.

CFI, [8]

Fault handling interrupt for corrected errors enable.

The fault handling interrupt is generated when one of the standard CE counters on ERR0MISC0 overflows and the overflow bit is set. The possible values are:

`0` Fault handling interrupt not generated for corrected errors.

`1` Fault handling interrupt generated for corrected errors.

The interrupt is generated even if the error status is overwritten because the error record already records a higher priority error.

——— **Note** ———

This applies to both reads and writes.

———————————

RES0, [7:4]

RES0 Reserved.

FI, [3]

Fault handling interrupt enable.

The fault handling interrupt is generated for all detected Deferred errors and Uncorrected errors. The possible values are:

`0` Fault handling interrupt disabled.

`1` Fault handling interrupt enabled.

UI, [2]

Uncorrected error recovery interrupt enable. When enabled, the error recovery interrupt is generated for all detected Uncorrected errors that are not deferred. The possible values are:

`0` Error recovery interrupt disabled.

`1` Error recovery interrupt enabled.

Note

Applies to both reads and writes.

RES0, [1]

RES0 Reserved.

ED, [0]

This bit is RES0. This control is not implemented in the Cortex-A75 core, error detection is always enabled.

Configurations

This register is accessible from the following registers when ERRSELR.SEL==0:

- [31:0]: *B1.32 ERXCTLR, Selected Error Record Control Register* on page B1-180
- [63:32]: *B1.33 ERXCTLR2, Selected Error Record Control Register 2* on page B1-181.
- *B2.39 ERXCTLR_EL1, Selected Error Record Control Register, EL1* on page B2-342.

B3.4 ERR0FR, Error Record Feature Register

The ERR0FR defines which of the common architecturally defined features are implemented and, of the implemented features, which are software programmable.

Bit field descriptions

ERR0FR is a 64-bit register, and is part of the *Reliability, Availability, Serviceability* (RAS) registers functional group.

The register is Read Only.

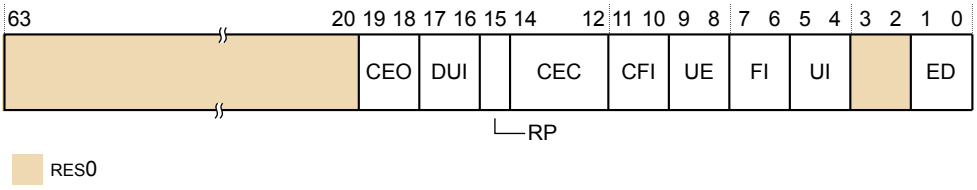

Figure B3-3 ERR0FR bit assignments

[63:20]

Reserved, RES0.

CEO, [19:18]

Corrected Error Overwrite. The value is:

`0b00` Counts CE if a counter is implemented and keeps the previous error status. If the counter overflows, ERR0STATUS.OF is set to 1.

DUI, [17:16]

Error recovery interrupt for deferred errors. The value is:

`0b00` The core does not support this feature.

RP, [15]

Repeat counter. The value is:

`0b1` A first repeat counter and a second other counter are implemented. The repeat counter is the same size as the primary error counter.

CEC, [14:12]

Corrected Error Counter. The value is:

`0b010` The node implements an 8-bit standard CE counter in ERR0MISC0[39:32].

CFI, [11:10]

Fault handling interrupt for corrected errors. The value is:

`0b10` The node implements a control for enabling fault handling interrupts on corrected errors.

UE, [9:8]

In-band uncorrected error reporting. The value is:

`0b01` The node implements in-band uncorrected error reporting, that is external aborts.

FI, [7:6]

Fault handling interrupt. The value is:

`0b10` The node implements a fault handling interrupt and implements controls for enabling and disabling.

UI, [5:4]

Error recovery interrupt for uncorrected errors. The value is:

`0b10` The node implements an error recovery interrupt and implements controls for enabling and disabling.

[3:2]

Reserved, RES0.

ED, [1:0]

Error detection and correction. The value is:

`0b01` The node always enables error detection and correction.

Configurations

ERR0FR resets to `0x000000000000A9A1`

ERR0FR is accessible from the following registers when ERRSELR.SEL==0:

- [31:0]: *B1.34 ERXFR, Selected Error Record Feature Register* on page B1-182.
- [63:32]: *B1.35 ERXFR2, Selected Error Record Feature Register 2* on page B1-183.
- *B2.40 ERXFR_EL1, Selected Error Record Feature Register, EL1* on page B2-343.

B3.5 ERR0MISC0, Error Record Miscellaneous Register 0

The ERR0MISC0 is an error syndrome register. It contains corrected error counters, information to identify where the error was detected, and other state information not present in the corresponding status and address error record registers.

Bit field descriptions

ERR0MISC0 is a 64-bit register, and is part of the *Reliability, Availability, Serviceability* (RAS) registers functional group.

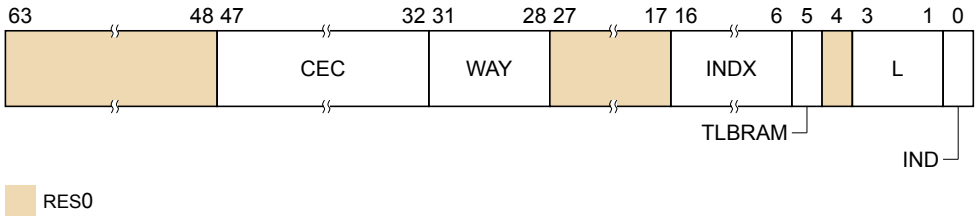

Figure B3-4 ERR0MISC0 bit assignments

[63:48]

Reserved, RES0.

CEC, [47:32]

Corrected error count.

Two corrected error counters are implemented.

WAY, [31:28]

Indicates the way that contained the error.

- For the L1 data RAM, all four bits are used.
- For the TLB and L1 instruction RAMs, only bits[31:30] are used.
- For the L2 RAM only bits[31:29] are used.

[27:17]

Reserved, RES0.

INDX, [16:6]

Indicates the index that contained the error.

Upper bits of the index are unused depending on the cache size.

TLBRAM, [5]

When an error occurs on a TLB RAM, this bit indicates in which TLB RAM block the error occurs. The possible values are:

`0` RAM0, small RAM.

`1` RAM1, large RAM.

[4]

Reserved, RES0.

L, [3:1]

Indicates the level that contained the error. The possible values are:

`0b000` Level 1.

`0b001` Level 2.

IND, [0]

Indicates the type of cache that contained the error. The possible values are:

`0` L1 data cache, unified L2 cache, or TLB.

`1` L1 instruction cache.

Configurations

ERR0MISC0 resets to [63:32] is `0x00000000`, [31:0] is UNKNOWN.

This register is accessible from the following registers when ERRSELR.SEL==0:

- [31:0]: *B1.36 ERXMISC0, Selected Error Miscellaneous Register 0* on page B1-184.
- [63:32]: *B1.37 ERXMISC1, Selected Error Miscellaneous Register 1* on page B1-185.
- *B2.41 ERXMISC0_EL1, Selected Error Record Miscellaneous Register 0, EL1* on page B2-344.

B3.6 ERR0MISC1, Error Record Miscellaneous Register 1

This register is unused in the Cortex-A75 core and marked as RES0.

Configurations

ERR0MISC1 is accessible from the following registers when ERRSELR.SEL==0:

- [31:0]: *B1.38 ERXMISC2, Selected Error Record Miscellaneous Register 2* on page B1-186.
- [63:32]: *B1.39 ERXMISC3, Selected Error Record Miscellaneous Register 3* on page B1-187.
- *B2.42 ERXMISC1_EL1, Selected Error Record Miscellaneous Register 1, EL1* on page B2-345.

B3.7 ERR0PFGCDNR, Error Pseudo Fault Generation Count Down Register

ERR0PFGCDNR is the Cortex-A75 node register that generates one of the errors that are enabled in the corresponding ERR0PFGCTL register.

Bit field descriptions

ERR0PFGCDNR is a 32-bit read/write register.

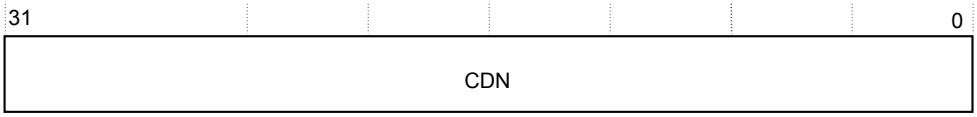

Figure B3-5 ERR0PFGCDNR bit assignments

CDN, [31:0]
Count Down value. The reset value of the Error Generation Counter is used for the countdown.

Configurations
There are no configuration options.

ERR0PFGCDNR resets to UNKNOWN.

ERR0PFGCDNR is accessible from the following registers when ERRSELR.SEL==0:

- *B1.40 ERXPFGCDNR, Selected Error Pseudo Fault Generation Count Down Register* on page B1-188.
- *B2.43 ERXPFGCDNR_EL1, Selected Error Pseudo Fault Generation Count Down Register, EL1* on page B2-346.

B3.8 ERR0PFGCTLR, Error Pseudo Fault Generation Control Register

The ERR0PFGCTLR is the Cortex-A75 node register that enables controlled fault generation.

Bit field descriptions

ERR0PFGCTLR is a 32-bit read/write register.

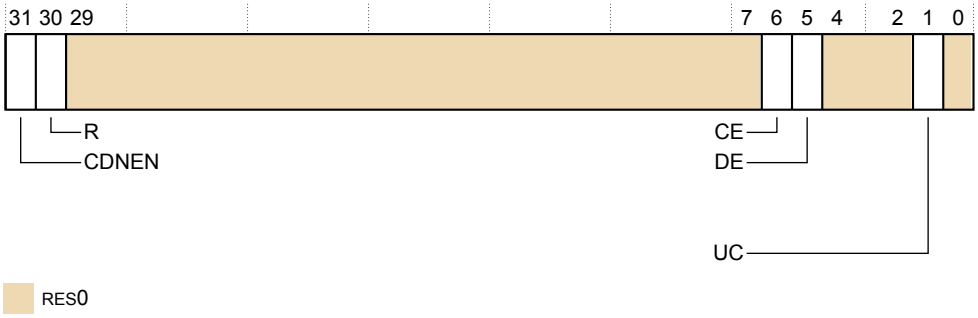

Figure B3-6 ERR0PFGCTLR bit assignments

CDNEN, [31]
Count down enable. This bit controls transfers from the value that is held in the ERR0PFGCDNR into the Error Generation Counter and enables this counter to start counting down. The possible values are:

`0` The Error Generation Counter is disabled.

`1` The value that is held in the ERR0PFGCDNR register is transferred into the Error Generation Counter. The Error Generation Counter counts down.

R, [30]
Restartable bit. When it reaches 0, the Error Generation Counter restarts from the ERR0PFGCDNR value or stops. The possible values are:

`0` When it reaches 0, the counter stops.

`1` When it reaches 0, the counter reloads the value that is stored in ERR0PFGCDNR and starts counting down again.

[29:7]
Reserved, RES0.

CE, [6]
Corrected error generation enable. The possible values are:

`0` No corrected error is generated.

`1` A corrected error might be generated when the Error Generation Counter is triggered.

DE, [5]
Deferred Error generation enable. The possible values are:

`0` No deferred error is generated.

`1` A deferred error might be generated when the Error Generation Counter is triggered.

[4:2]
Reserved, RES0.

UC, [1]
Uncontainable error generation enable. The possible values are:

`0` No uncontainable error is generated.

`1` An uncontainable error might be generated when the Error Generation Counter is triggered.

[0]

Reserved, RES0.

Configurations

There are no configuration notes.

ERR0PFGCTLR resets to `0x00000000`.

ERR0PFGCTLR is accessible from the following registers when ERRSELR.SEL==0:

- *B1.41 ERXPFGCTLR, Selected Error Pseudo Fault Generation Control Register* on page B1-190.
- *B2.44 ERXPFGCTLR_EL1, Selected Error Pseudo Fault Generation Control Register, EL1* on page B2-348.

B3.9 ERR0PFGFR, Error Pseudo Fault Generation Feature Register

The ERR0PFGFR is the Cortex-A75 node register that defines which fault generation features are implemented.

Bit field descriptions

ERR0PFGFR is a 32-bit read-only register.

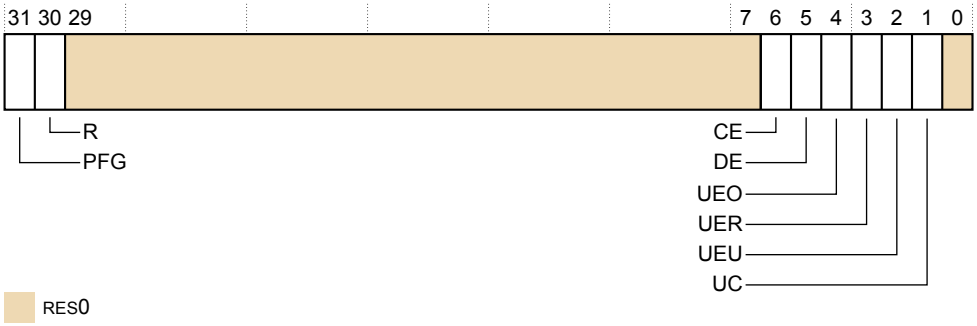

Figure B3-7 ERR0PFGFR bit assignments

PFG, [31]
Pseudo Fault Generation. The value is:

`1` The node implements a fault injection mechanism.

R, [30]
Restartable bit. When it reaches zero, the Error Generation Counter restarts from the ERR0PFGCDN value or stops. The value is:

`1` This feature is controllable.

[29:7]
Reserved, RES0.

CE, [6]
Corrected Error generation. The value is:

`1` This feature is controllable.

DE, [5]
Deferred Error generation. The value is:

`1` This feature is controllable.

UEO, [4]
Latent or Restartable Error generation. The value is:

`0` The node does not support this feature.

UER, [3]
Signaled or Recoverable Error generation. The value is:

`0` The node does not support this feature.

UEU, [2]
Unrecoverable Error generation. The value is:

`0` The node does not support this feature.

RES0, [2]
Reserved, RES0.

UC, [1]

Uncontainable Error generation. The value is:

`1` This feature is controllable.

[0]

Reserved, RES0.

Configurations

There are no configuration notes.

ERR0PFGFR resets to `0xC0000062`.

ERR0PFGFR is accessible from the following registers when ERRSELR.SEL==0:

- *B1.42 ERXPFGFR, Selected Pseudo Fault Generation Feature Register* on page B1-192.
- *B2.45 ERXPFGFR_EL1, Selected Pseudo Fault Generation Feature Register, EL1* on page B2-350.

B3.10 ERR0STATUS, Error Record Primary Status Register

The ERR0STATUS contains information about the error record:

- Whether any error has been detected.
- Whether any detected error was not corrected and returned to a master.
- Whether any detected error was not corrected and deferred.
- Whether a second error of the same type was detected before software handled the first error.
- Whether any error has been reported.
- Whether the other error record registers contain valid information.

Bit field descriptions

ERR0STATUS is a 32-bit register.

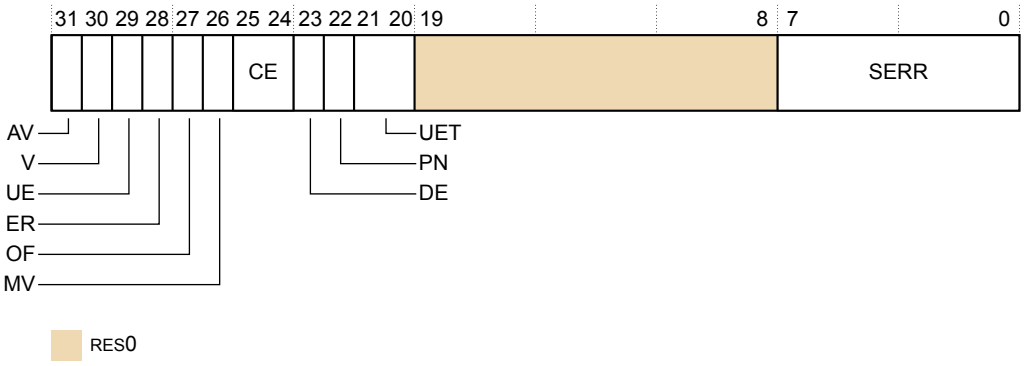

Figure B3-8 ERR0STATUS bit assignments

OF, [27]

Overflow. The possible values are:

`0` • If UE == 1, then no error status for an Uncorrected error has been discarded.
• If UE == 0 and DE == 1, then no error status for a Deferred error has been discarded.
• If UE == 0, DE == 0, and CE !== `0b00`, then: The corrected error counter has not overflowed.

`1` More than one error has occurred and so details of the other error have been discarded.

MV, [26]

Miscellaneous Registers Valid. The possible values are:

`0` ERR0MISC0 and ERR0MISC1 are not valid.

`1` This bit indicates that ERR0MISC0 contains additional information about any error that is recorded by this record.

CE, [25:24]

Corrected error. The possible values are:

`0b00` No corrected error recorded.

`0b10` At least one corrected error recorded.

DE, [23]

Deferred error. The possible values are:

`0` No errors were deferred.

`1` At least one error was not corrected and deferred by poisoning.

PN, [22]

Poison. The value is:

`0` The Cortex-A75 core cannot distinguish a poisoned value from a corrupted value.

UET, [21:20]

Uncorrected Error Type. The value is:

`0b00` Uncontainable.

[19:8]

RES0.

Reserved.

SERR, [7:0]

Primary error code. The possible values are:

`0x0` No error.

`0x2` ECC error from internal data buffer.

`0x6` ECC error on cache data RAM.

`0x7` ECC error on cache tag or dirty RAM.

`0x8` Parity error on TLB data RAM.

`0x9` Parity error on TLB tag RAM.

`0x15` Deferred error from slave not supported at the consumer. For example, poisoned data received from a slave by a master that cannot defer the error further.

Configurations

There are no configuration notes.

ERR0STATUS resets to `0x00000000`.

ERR0STATUS is accessible from the following registers when ERRSELR.SEL==0:

- *B1.43 ERXSTATUS, Selected Error Record Primary Status Register* on page B1-193.
- *B2.46 ERXSTATUS_EL1, Selected Error Record Primary Status Register, EL1* on page B2-351.

Chapter B4
GIC registers

This chapter describes the GIC registers.

It contains the following sections:

B4.1 CPU interface registers

Each CPU interface block provides the interface for the Cortex-A75 core that interfaces with a GIC distributor within the system.

The Cortex-A75 core only supports system register access to the GIC CPU interface registers. The following table lists the three types of GIC CPU interface system registers supported in the Cortex-A75 core.

Table B4-1 GIC CPU interface system register types supported in the Cortex-A75 core.

Register prefix	Register type
ICC	Physical GIC CPU interface system registers.
ICV	Virtual GIC CPU interface system registers.
ICH	Virtual interface control system registers.

Access to virtual GIC CPU interface system registers is only possible at Non-secure EL1.

Access to ICC registers or the equivalent ICV registers is determined by HCR_EL2. See *B2.51 HCR_EL2, Hypervisor Configuration Register, EL2* on page B2-356.

For more information on the CPU interface, see the *Arm® Generic Interrupt Controller Architecture Specification*.

B4.2 AArch32 physical GIC CPU interface system register summary

The following table lists the AArch32 physical GIC CPU interface system registers that have implementation-defined bits.

See the *Arm® Generic Interrupt Controller Architecture Specification* for more information and a complete list of AArch32 physical GIC CPU interface system registers.

Table B4-2 AArch32 physical GIC CPU interface system register summary

Name	Op1	CRn	CRm	Op2	Type	Description
ICC_AP0R0	0	12	8	4	RW	*B4.3 ICC_AP0R0, Interrupt Controller Active Priorities Group 0 Register 0* on page B4-459
ICC_AP1R0	0	12	9	0	RW	*B4.4 ICC_AP1R0, Interrupt Controller Active Priorities Group 1 Register 0* on page B4-460
ICC_BPR0	0	12	8	3	RW	*B4.5 ICC_BPR0, Interrupt Controller Binary Point Register 0* on page B4-461
ICC_BPR1	0	12	12	3	RW	*B4.6 ICC_BPR1, Interrupt Controller Binary Point Register 1* on page B4-462
ICC_CTLR	0	12	12	4	RW	*B4.7 ICC_CTLR, Interrupt Controller Control Register* on page B4-463
ICC_HSRE	4	12	9	5	RW	*B4.8 ICC_HSRE, Interrupt Controller Hyp System Register Enable Register* on page B4-465
ICC_MCTLR	6	12	12	4	RW	*B4.9 ICC_MCTLR, Interrupt Controller Monitor Control Register* on page B4-467
ICC_MSRE	6	12	12	5	RW	*B4.10 ICC_MSRE, Interrupt Controller Monitor System Register Enable Register* on page B4-469
ICC_SRE	0	12	12	5	RW	*B4.11 ICC_SRE, Interrupt Controller System Register Enable Register* on page B4-470

B4.3 ICC_AP0R0, Interrupt Controller Active Priorities Group 0 Register 0

The ICC_AP0R0 provides information about Group 0 active priorities.

Bit field descriptions

This register is a 32-bit register and is part of:

- The GIC system registers functional group.
- The GIC control registers functional group.

The core implements 5 bits of priority with 32 priority levels, corresponding to the 32 bits [31:0] of the register. The possible values for each bit are:

`0x00000000` No interrupt active. This is the reset value.

`0x00000001` Interrupt active for priority `0x0`.

`0x00000002` Interrupt active for priority `0x8`.

...

`0x80000000` Interrupt active for priority `0xF8`.

Configurations

There is one instance of this register that is used in both Secure and Non-secure states.

AArch32 System register ICC_AP0R0 is architecturally mapped to AArch64 System register ICC_AP0R0_EL1.

Details not provided in this description are architecturally defined. See the *Arm® Generic Interrupt Controller Architecture Specification*.

B4.4 ICC_AP1R0, Interrupt Controller Active Priorities Group 1 Register 0

The ICC_AP1R0 provides information about Group 1 active priorities.

Bit field descriptions

This register is a 32-bit register and is part of:

- The GIC system registers functional group.
- The GIC control registers functional group.

The core implements 5 bits of priority with 32 priority levels, corresponding to the 32 bits [31:0] of the register. The possible values for each bit are:

`0x00000000` No interrupt active. This is the reset value.

`0x00000001` Interrupt active for priority `0x0`.

`0x00000002` Interrupt active for priority `0x8`.

...

`0x80000000` Interrupt active for priority `0xF8`.

Configurations

There is one instance of this register that is used in both Secure and Non-secure states.

AArch32 System register ICC_AP1R0 is architecturally mapped to AArch64 System register ICC_AP1R0_EL1.

Details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

B4.5 ICC_BPR0, Interrupt Controller Binary Point Register 0

ICC_BPR0 defines the point at which the priority value fields split into two parts, the group priority field and the subpriority field. The group priority field determines Group 0 interrupt preemption.

Bit field descriptions

ICC_BPR0 is a 32-bit register and is part of:

- The GIC system registers functional group.
- The GIC control registers functional group.

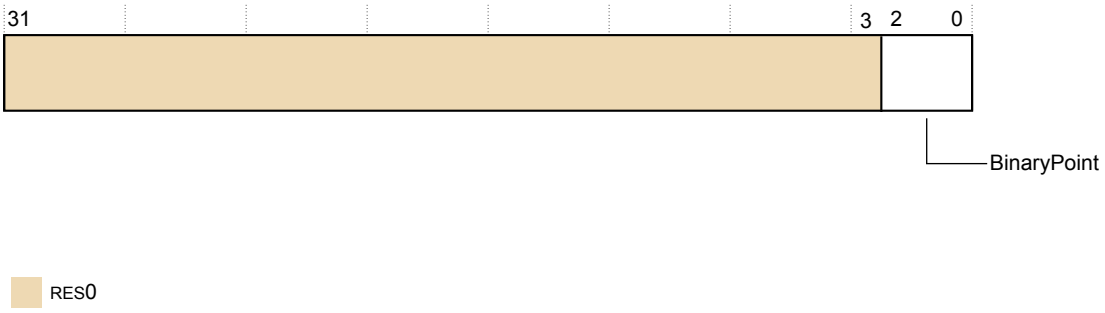

Figure B4-1 ICC_BPR0 bit assignments

RES0, [31:3]

Reserved, RES0.

BinaryPoint, [2:0]

The value of this field controls how the 8-bit interrupt priority field is split into a group priority field, that determines interrupt preemption, and a subpriority field. The minimum value implemented is:

`0x2`

Configurations

AArch32 System register ICC_BPR0 is architecturally mapped to AArch64 System register ICC_BPR0_EL1.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Generic Interrupt Controller Architecture Specification*.

B4.6 ICC_BPR1, Interrupt Controller Binary Point Register 1

ICC_BPR1 defines the point at which the priority value fields split into two parts, the group priority field and the subpriority field. The group priority field determines Group 1 interrupt preemption.

Bit field descriptions

ICC_BPR1 is a 32-bit register and is part of:

- The GIC system registers functional group.
- The GIC control registers functional group.

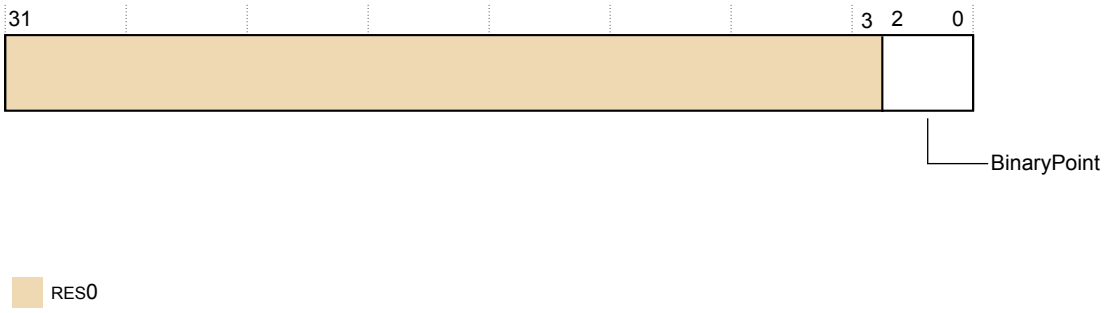

Figure B4-2 ICC_BPR1 bit assignments

RES0, [31:3]

Reserved, RES0.

BinaryPoint, [2:0]

The value of this field controls how the 8-bit interrupt priority field is split into a group priority field, that determines interrupt preemption, and a subpriority field.

The minimum value implemented of ICC_BPR1_EL1 Secure register is `0x2`.

The minimum value implemented of ICC_BPR1_EL1 Non-secure register is `0x3`.

Configurations

AArch32 System register ICC_BPR1 is architecturally mapped to AArch64 System register ICC_BPR1_EL1.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Generic Interrupt Controller Architecture Specification*.

B4.7 ICC_CTLR, Interrupt Controller Control Register

ICC_CTLR controls aspects of the behavior of the GIC CPU interface and provides information about the features implemented.

Bit field descriptions

ICC_CTLR is a 32-bit register and is part of:

- The GIC system registers functional group.
- The GIC control registers functional group.

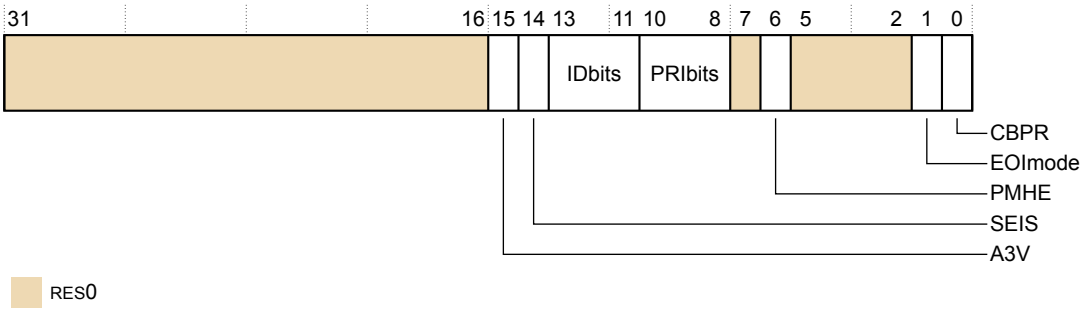

Figure B4-3 ICC_CTLR bit assignments

RES0, [31:16]

Reserved, RES0.

A3V, [15]

Affinity 3 Valid. The value is:

`0x1` The CPU interface logic supports non-zero values of Affinity 3 in SGI generation System registers.

SEIS, [14]

SEI Support.The value is:

`0x0` The CPU interface logic does not support local generation of SEIs.

IDbits, [13:11]

Identifier bits. The value is:

`0x0` The number of physical interrupt identifier bits supported is 16 bits.

This field is an alias of ICC_CTLR_EL3.IDbits.

PRIbits, [10:8]

Priority bits. The value is:

`0x4` The core support 32 levels of physical priority with 5 priority bits.

RES0, [7]

Reserved, RES0.

PMHE, [6]

Priority Mask Hint Enable. This bit is an alias of ICC_CTLR_EL3.PMHE. The possible values are:

0 Disables use of ICC_PMR as a hint for interrupt distribution.

1 Enables use of ICC_PMR as a hint for interrupt distribution.

RES0, [5:2]

Reserved, RES0.

EOImode, [1]

End of interrupt mode for the current security state. The possible values are:

0 ICC_EOIR0 and ICC_EOIR1 provide both priority drop and interrupt deactivation functionality. Accesses to ICC_DIR are UNPREDICTABLE.

1 ICC_EOIR0 and ICC_EOIR1 provide priority drop functionality only. ICC_DIR provides interrupt deactivation functionality.

If EL3 is using AArch32, this bit is an alias of ICC_MCTLR.EOImode_EL1 {S, NS}.

If EL3 is using AArch64, this bit is an alias of ICC_CTLR_EL3.EOImode_EL1 {S, NS}.

CBPR, [0]

Common Binary Point Register. Control whether the same register is used for interrupt preemption of both Group 0 and Group 1 interrupt. The possible values are:

0 ICC_BPR0 determines the preemption group for Group 0 interrupts.

ICC_BPR1 determines the preemption group for Group 1 interrupts.

1 ICC_BPR0 determines the preemption group for Group 0 and Group 1 interrupts.

If EL3 is using AArch32, this bit is an alias of ICC_MCTLR.CBPR_EL1 {N, NS}.

If EL3 is using AArch64, this bit is an alias of ICC_CTLR_EL3.CBPR_EL1 {S, NS}.

If GICD_CTLR.DS == 0, this bit is read-only.

If GICD_CTLR.DS == 0, this bit is read/write.

Configurations

AArch32 System register ICC_CTLR (S) is architecturally mapped to AArch64 System register ICC_CTLR_EL1 (S).

AArch32 System register ICC_CTLR (NS) is architecturally mapped to AArch64 System register ICC_CTLR_EL1(NS).

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Generic Interrupt Controller Architecture Specification*.

B4.8 ICC_HSRE, Interrupt Controller Hyp System Register Enable Register

ICC_HSRE controls whether the System register interface or the memory-mapped interface to the GIC CPU interface is used for EL2.

Bit field descriptions

ICC_HSRE is a 32-bit register and is part of:

- The GIC system registers functional group.
- The Virtualization registers functional group.
- The GIC control registers functional group.

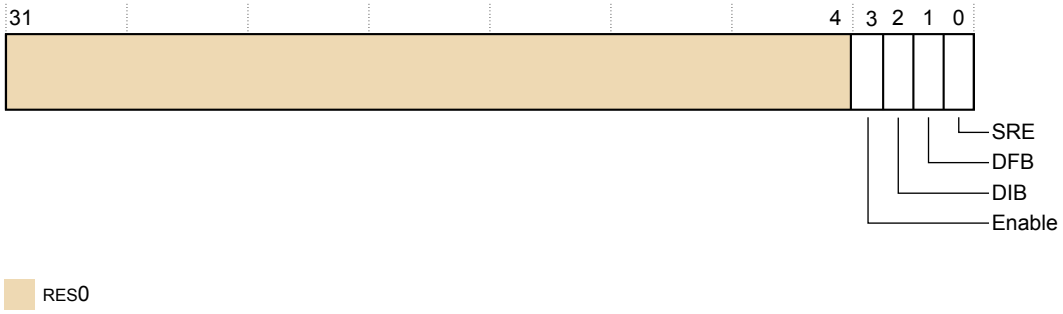

Figure B4-4 ICC_HSRE bit assignments

RES0, [31:4]

Reserved, RES0.

Enable, [3]

Enables lower Exception level access to ICC_SRE. The value is:

`0x1` Non-secure EL1 accesses to ICC_SRE do not trap to EL2.

This bit is RAO/WI.

DIB, [2]

Disable IRQ bypass. The possible values are:

`0x0` IRQ bypass enabled.

`0x1` IRQ bypass disabled.

This bit is an alias of ICC_MSRE.DIB.

DFB, [1]

Disable FIQ bypass. The possible values are:

`0x0` FIQ bypass enabled.

`0x1` FIQ bypass disabled.

This bit is an alias of ICC_MSRE.DFB.

SRE, [0]

System Register Enable. The value is:

`0x1` The System register interface for the current Security state is enabled.

This bit is RAO/WI. The core only supports a system register interface to the GIC CPU interface.

Configurations

AArch32 System register ICC_HSRE (S) is architecturally mapped to AArch64 System register ICC_SRE_EL2 (S).

AArch32 System register ICC_HSRE (NS) is architecturally mapped to AArch64 System register ICC_SRE_EL2 (NS).

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Generic Interrupt Controller Architecture Specification*.

B4.9 ICC_MCTLR, Interrupt Controller Monitor Control Register

ICC_MCTLR controls aspects of the behavior of the GIC CPU interface and provides information about the features implemented.

Bit field descriptions

ICC_MCTLR is a 32-bit register and is part of:

- The GIC system registers functional group.
- The Security registers functional group.
- The GIC control registers functional group.

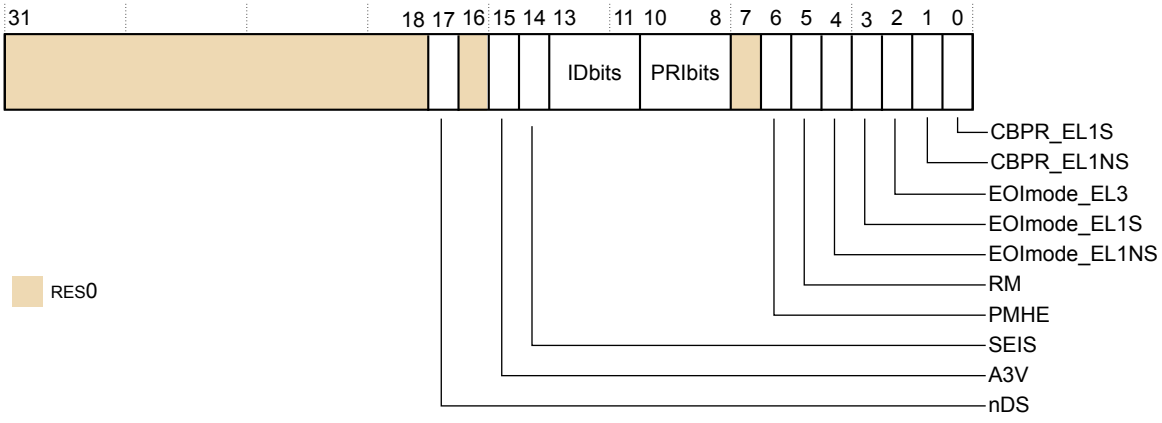

Figure B4-5 ICC_MCTLR bit assignments

RES0, [31:18]

Reserved, RES0.

nDS, [17]

Disable Security not supported. Read-only and writes are ignored. The value is:

`0x1` The CPU interface logic does not support disabling of security, and requires that security is not disabled.

RES0, [16]

Reserved, RES0.

A3V, [15]

Affinity 3 Valid. The value is:

`0x1` The CPU interface logic supports non-zero values of the Aff3 field in SGI generation System registers.

SEIS, [14]

SEI Support. The value is:

`0x0` The CPU interface logic does not support generation of SEIs.

IDbits, [13:11]

Identifier bits. The value is:

`0x0` The number of physical interrupt identifier bits supported is 16 bits.

This field is an alias of ICC_CTLR_EL3.IDbits.

PRIbits, [10:8]

Priority bits. The value is:

`0x4` The core support 32 levels of physical priority with 5 priority bits.

RES0, [7]

Reserved, RES0.

PMHE, [6]

Priority Mask Hint Enable.

RM, [5]

SBZ. The equivalent bit in AArch64 is the Routing Modifier bit. This feature is not supported when EL3 is using AArch32. The value is:

`0x0`

EOImode_EL1NS, [4]

EOI mode for interrupts handled at Non-secure EL1 and EL2. Controls whether a write to an End of Interrupt register also deactivates the interrupt.

EOI mode for interrupts handled at Non-secure EL1 and EL2.

EOImode_EL1S, [3]

EOI mode for interrupts handled at Secure EL1. Controls whether a write to an End of Interrupt register also deactivates the interrupt.

EOI mode for interrupts handled at Secure EL1

EOImode_EL3, [2]

EOI mode for interrupts handled at EL3. Controls whether a write to an End of Interrupt register also deactivates the interrupt.

EOI mode for interrupts handled at EL3.

CBPR_EL1NS, [1]

Common Binary Point Register, EL1 Non-secure.

Control whether the same register is used for interrupt pre-emption of both Group 0 and Group 1 Non-secure interrupts at EL1 and EL2.

CBPR_EL1S, [0]

Common Binary Point Register, EL1 Secure.

Control whether the same register is used for interrupt pre-emption of both Group 0 and Group 1 Secure interrupt at EL1.

Configurations

This register is only accessible in Secure state.

AArch32 System register ICC_MCTLR can be mapped to AArch64 System register ICC_CTLR_EL3.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Generic Interrupt Controller Architecture Specification*.

B4.10 ICC_MSRE, Interrupt Controller Monitor System Register Enable Register

ICC_MSRE controls whether the System register interface or the memory-mapped interface to the GIC CPU interface is used for EL3.

Bit field descriptions

ICC_MSRE is a 32-bit register and is part of:

- The GIC system registers functional group.
- The Security registers functional group.
- The GIC control registers functional group.

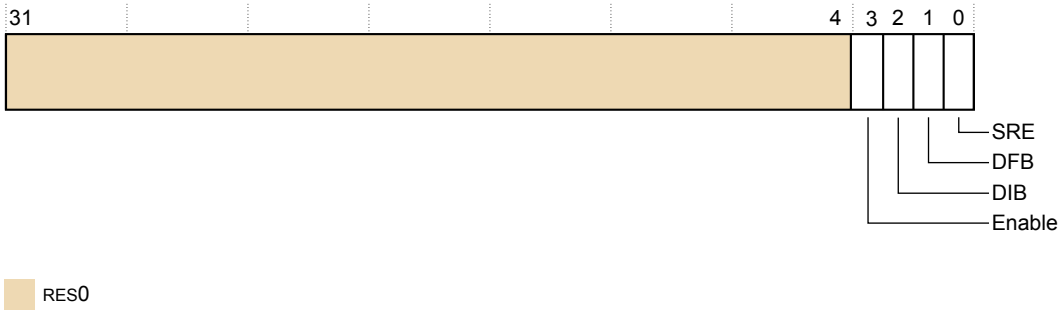

Figure B4-6 ICC_MSRE bit assignments

RES0, [31:4]

Reserved, RES0.

Enable, [3]

Enables lower Exception level access to ICC_SRE. The value is:

`0x1` Non-secure EL1 accesses to ICC_SRE do not trap to EL2.

This bit is RAO/WI.

DIB, [2]

Disable IRQ bypass. The possible values are:

`0x0` IRQ bypass enabled.

`0x1` IRQ bypass disabled.

DFB, [1]

Disable FIQ bypass. The possible values are:

`0x0` FIQ bypass enabled.

`0x1` FIQ bypass disabled.

SRE, [0]

System Register Enable. The value is:

`0x1` The System register interface for the current Security state is enabled.

This bit is RAO/WI. The core only supports a system register interface to the GIC CPU interface.

Configurations

AArch32 System register ICC_MSRE can be mapped to AArch64 System register ICC_SRE_EL3.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Generic Interrupt Controller Architecture Specification*.

B4.11 ICC_SRE, Interrupt Controller System Register Enable Register

ICC_SRE controls whether the System register interface or the memory-mapped interface to the GIC CPU interface is used for EL0 and EL1.

Bit field descriptions

ICC_SRE is a 32-bit register and is part of:

- The GIC system registers functional group.
- The GIC control registers functional group.

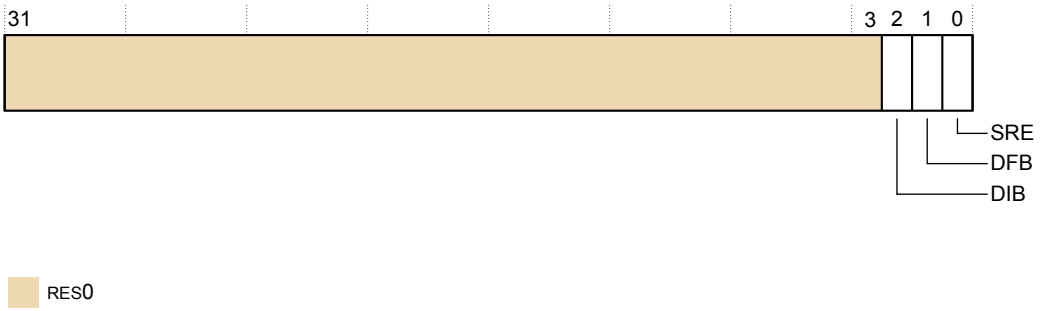

Figure B4-7 ICC_SRE bit assignments

RES0, [31:3]

Reserved, RES0.

DIB, [2]

Disable IRQ bypass. The possible values are:

`0x0` IRQ bypass enabled.

`0x1` IRQ bypass disabled.

This bit is an alias of ICC_MSRE.DIB.

DFB, [1]

Disable FIQ bypass. The possible values are:

`0x0` FIQ bypass enabled.

`0x1` FIQ bypass disabled.

This bit is an alias of ICC_MSRE.DFB.

SRE, [0]

System Register Enable. The value is:

`0x1` The System register interface for the current Security state is enabled.

This bit is RAO/WI. The core only supports a system register interface to the GIC CPU interface.

Configurations

AArch32 System register ICC_SRE (S) is architecturally mapped to AArch64 System register ICC_SRE_EL1 (S).

AArch32 System register ICC_SRE (NS) is architecturally mapped to AArch64 System register ICC_SRE_EL1(NS).

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Generic Interrupt Controller Architecture Specification*.

B4.12 AArch32 virtual GIC CPU interface register summary

The following table describes the AArch32 virtual GIC CPU interface system register that has implementation defined bits.

See the *Arm® Generic Interrupt Controller Architecture Specification* for more information and a complete list of AArch32 virtual GIC CPU interface system registers.

Table B4-3 AArch32 virtual GIC CPU interface register summary

Name	Op1	CRn	CRm	Op2	Type	Description
ICV_AP0R0	0	12	8	4	RW	*B4.13 ICV_AP0R0, Interrupt Controller Virtual Active Priorities Group 0 Register 0* on page B4-472
ICV_AP1R0	0	12	9	0	RW	*B4.14 ICV_AP1R0, Interrupt Controller Virtual Active Priorities Group 1 Register 0* on page B4-473
ICV_BPR0	0	12	8	3	RW	*B4.15 ICV_BPR0, Interrupt Controller Virtual Binary Point Register 0* on page B4-474
ICV_BPR1	0	12	12	3	RW	*B4.16 ICV_BPR1, Interrupt Controller Virtual Binary Point Register 1* on page B4-475
ICV_CTLR	0	12	12	4	RW	*B4.17 ICV_CTLR, Interrupt Controller Virtual Control Register* on page B4-476

B4.13 ICV_AP0R0, Interrupt Controller Virtual Active Priorities Group 0 Register 0

The ICV_AP0R0 register provides information about virtual Group 0 active priorities.

Bit descriptions

This register is a 32-bit register and is part of:

- The GIC system registers functional group.
- The GIC control registers functional group.

The core implements 5 bits of priority with 32 priority levels, corresponding to the 32 bits [31:0] of the register. The possible values for each bit are:

`0x00000000` No interrupt active. This is the reset value.

`0x00000001` Interrupt active for priority `0x0`.

`0x00000002` Interrupt active for priority `0x8`.

...

`0x80000000` Interrupt active for priority `0xF8`.

Configurations

AArch32 System register ICV_AP0R0 is architecturally mapped to AArch64 System register ICV_AP0R0_EL1.

Details not provided in this description are architecturally defined. See the *Arm® Generic Interrupt Controller Architecture Specification*.

B4.14 ICV_AP1R0, Interrupt Controller Virtual Active Priorities Group 1 Register 0

The ICV_AP1R0 register provides information about virtual Group 0 active priorities.

Bit descriptions

This register is a 32-bit register and is part of:

- The GIC system registers functional group.
- The GIC control registers functional group.

The core implements 5 bits of priority with 32 priority levels, corresponding to the 32 bits [31:0] of the register. The possible values for each bit are:

`0x00000000` No interrupt active. This is the reset value.

`0x00000001` Interrupt active for priority `0x0`.

`0x00000002` Interrupt active for priority `0x8`.

...

`0x80000000` Interrupt active for priority `0xF8`.

Configurations

AArch32 System register ICV_AP1R0 is architecturally mapped to AArch64 System register ICV_AP1R0_EL1.

Details not provided in this description are architecturally defined. See the *Arm® Generic Interrupt Controller Architecture Specification*.

B4.15 ICV_BPR0, Interrupt Controller Virtual Binary Point Register 0

ICV_BPR0 defines the point at which the priority value fields split into two parts, the group priority field and the subpriority field. The group priority field determines virtual Group 0 interrupt preemption.

Bit field descriptions

ICC_BPR0 is a 32-bit register and is part of:

- The GIC system registers functional group.
- The GIC virtual interface control registers functional group.

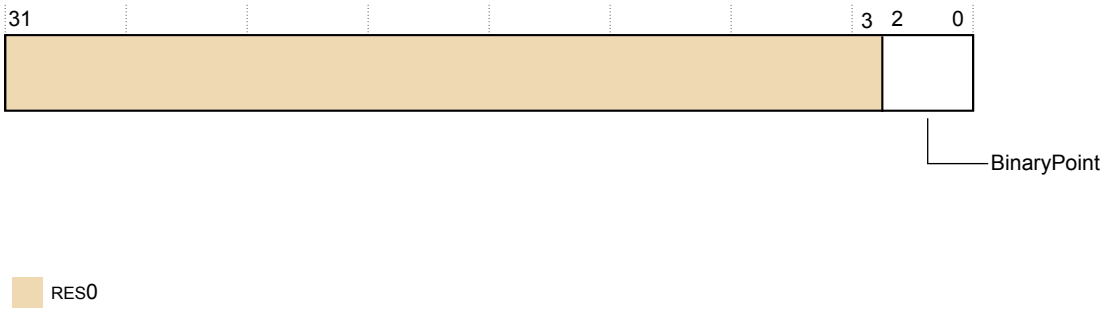

Figure B4-8 ICV_BPR0 bit assignments

RES0, [31:3]

Reserved, RES0.

BinaryPoint, [2:0]

The value of this field controls how the 8-bit interrupt priority field is split into a group priority field, that determines interrupt preemption, and a subpriority field. The minimum value implemented is:

`0x2`

Configurations

AArch32 System register ICV_BPR0 is architecturally mapped to AArch64 System register ICV_BPR0_EL1.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Generic Interrupt Controller Architecture Specification*.

B4.16 ICV_BPR1, Interrupt Controller Virtual Binary Point Register 1

ICV_BPR1 defines the point at which the priority value fields split into two parts, the group priority field and the subpriority field. The group priority field determines virtual Group 1 interrupt preemption.

Bit field descriptions

ICC_BPR1 is a 32-bit register and is part of:

- The GIC system registers functional group.
- The GIC virtual interface control registers functional group.

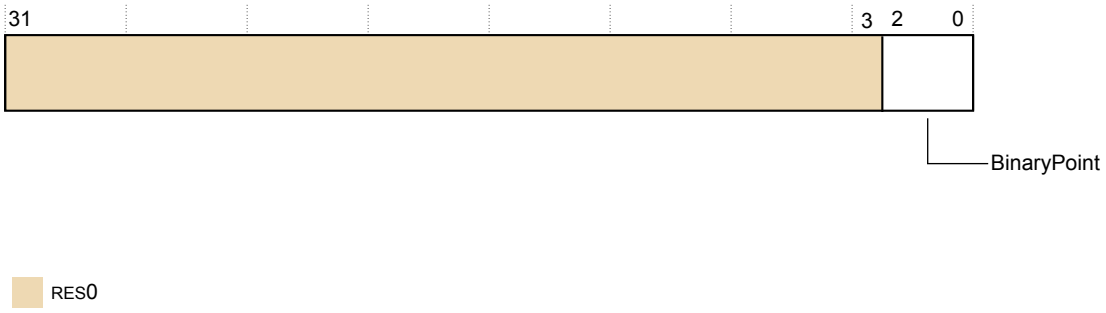

Figure B4-9 ICV_BPR1 bit assignments

RES0, [31:3]

Reserved, RES0.

BinaryPoint, [2:0]

The value of this field controls how the 8-bit interrupt priority field is split into a group priority field, that determines interrupt preemption, and a subpriority field.

The minimum value implemented of ICV_BPR1_EL1 Secure register is `0x2`.

The minimum value implemented of ICV_BPR1_EL1 Non-secure register is `0x3`.

Configurations

AArch32 System register ICV_BPR1 is architecturally mapped to AArch64 System register ICV_BPR1_EL1.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Generic Interrupt Controller Architecture Specification*.

B4.17 ICV_CTLR, Interrupt Controller Virtual Control Register

ICV_CTLR controls aspects of the behavior of the GIC virtual CPU interface and provides information about the features implemented.

Bit field descriptions

ICV_CTLR is a 32-bit register and is part of:

- The GIC system registers functional group.
- The GIC virtual interface control registers functional group.

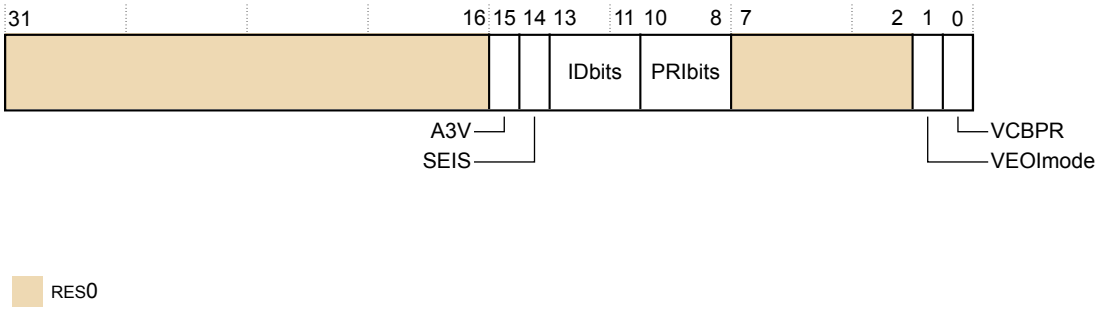

Figure B4-10 ICV_CTLR bit assignments

RES0, [31:16]

Reserved, RES0.

A3V, [15]

Affinity 3 Valid. The value is:

`0x1` The virtual CPU interface logic supports non-zero values of Affinity 3 in SGI generation System registers.

SEIS, [14]

SEI Support. The value is:

`0x0` The virtual CPU interface logic does not support local generation of SEIs.

IDbits, [13:11]

Identifier bits. The value is:

`0x0` The number of physical interrupt identifier bits supported is 16 bits.

PRIbits, [10:8]

Priority bits. The value is:

`0x4` Support 32 levels of physical priority (5 priority bits).

RES0, [7:2]

Reserved, RES0.

VEOImode, [1]

Virtual EOI mode. The possible values are:

`0x0` ICV_EOIR0 and ICV_EOIR1 provide both priority drop and interrupt deactivation functionality. Accesses to ICV_DIR are UNPREDICTABLE.

`0x1` ICV_EOIR0 and ICV_EOIR1 provide priority drop functionality only. ICV_DIR provides interrupt deactivation functionality.

VCBPR, [0]

Common Binary Point Register. Controls whether the same register is used for interrupt preemption of both virtual Group 0 and virtual Group 1 interrupts. The possible values are:

0 ICV_BPR0 determines the preemption group for virtual Group 0 interrupts only.

ICV_BPR1 determines the preemption group for virtual Group 1 interrupts.

1 ICV_BPR0 determines the preemption group for both virtual Group 0 and virtual Group 1 interrupts.

Reads of ICV_BPR1 return ICV_BPR0 plus one, saturated to 111. Writes to ICV_BPR1 are ignored.

Configurations

AArch32 System register ICV_CTLR is architecturally mapped to AArch64 System register ICV_CTLR_EL1.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Generic Interrupt Controller Architecture Specification*.

B4.18 AArch32 virtual interface control system register summary

The following table lists the AArch32 virtual interface control system registers that have implementation defined bits.

See the *Arm® Generic Interrupt Controller Architecture Specification* for more information and a complete list of AArch32 virtual interface control system registers.

Name	Op1	CRn	CRm	Op2	Type	Description
ICH_AP0R0	4	12	8	0	RW	*B4.19 ICH_AP0R0, Interrupt Controller Hyp Active Priorities Group 0 Register 0* on page B4-479
ICH_AP1R0	4	12	9	0	RW	*B4.20 ICH_AP1R0, Interrupt Controller Hyp Active Priorities Group 1 Register 0* on page B4-480
ICH_HCR	4	12	11	0	RW	*B4.21 ICH_HCR, Interrupt Controller Hyp Control Register* on page B4-481
ICH_LR0	4	12	12	0	RW	Interrupt Controller List Registers 0-3. The Cortex-A75 core implements four ICH_LR registers, as defined by ICH_VTR.ListRegs. Accesses to the rest of the ICH_LR registers are UNDEFINED.
ICH_LR1	4	12	12	1	RW	
ICH_LR2	4	12	12	2	RW	
ICH_LR3	4	12	12	3	RW	
ICH_VTR	4	12	11	1	RO	*B4.22 ICH_VMCR, Interrupt Controller Virtual Machine Control Register* on page B4-484
ICH_VMCR	4	12	11	7	RW	*B4.23 ICH_VTR, Interrupt Controller VGIC Type Register* on page B4-486

B4.19 ICH_AP0R0, Interrupt Controller Hyp Active Priorities Group 0 Register 0

The ICH_AP0R0 provides information about Group 0 active priorities for EL2.

Bit field descriptions

This register is a 32-bit register and is part of:

- The GIC system registers functional group.
- The Virtualization registers functional group.
- The GIC host interface control registers functional group.

The core implements 5 bits of priority with 32 priority levels, corresponding to the 32 bits [31:0] of the register. The possible values for each bit are:

`0x00000000` No interrupt active. This is the reset value.

`0x00000001` Interrupt active for priority `0x0`.

`0x00000002` Interrupt active for priority `0x8`.

...

`0x80000000` Interrupt active for priority `0xF8`.

Configurations

AArch32 System register ICH_AP0R0 is architecturally mapped to AArch64 System register ICH_AP0R0_EL2.

Details not provided in this description are architecturally defined. See the *Arm® Generic Interrupt Controller Architecture Specification*.

B4.20 ICH_AP1R0, Interrupt Controller Hyp Active Priorities Group 1 Register 0

The ICH_AP1R0 provides information about Group 1 active priorities for EL2.

Bit field descriptions

This register is a 32-bit register and is part of:

- The GIC system registers functional group.
- The Virtualization registers functional group.
- The GIC host interface control registers functional group.

The core implements 5 bits of priority with 32 priority levels, corresponding to the 32 bits [31:0] of the register. The possible values for each bit are:

`0x00000000` No interrupt active. This is the reset value.

`0x00000001` Interrupt active for priority `0x0`.

`0x00000002` Interrupt active for priority `0x8`.

...

`0x80000000` Interrupt active for priority `0xF8`.

Configurations

AArch32 System register ICH_AP1R0 is architecturally mapped to AArch64 System register ICH_AP1R0_EL2.

If EL2 is not implemented, this register is RES0 from EL3.

Details not provided in this description are architecturally defined. See the *Arm® Generic Interrupt Controller Architecture Specification*.

B4.21 ICH_HCR, Interrupt Controller Hyp Control Register

ICH_HCR controls the environment for VMs.

Bit field descriptions

ICH_HCR is a 32-bit register and is part of:

- The GIC system registers functional group.
- The Virtualization registers functional group.
- The GIC host interface control registers functional group.

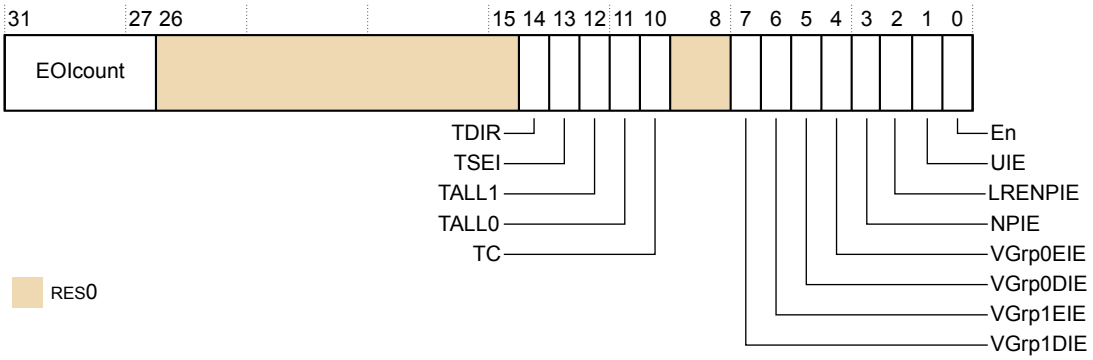

Figure B4-11 ICH_HCR bit assignments

EOIcount, [31:27]

Number of outstanding deactivates.

RES0, [26:15]

Reserved, RES0.

TDIR, [14]

Trap Non-secure EL1 writes to ICC_DIR and ICV_DIR. The possible values are:

`0x0` Non-secure EL1 writes of ICC_DIR and ICV_DIR are not trapped to EL2, unless trapped by other mechanisms.

`0x1` Non-secure EL1 writes of ICC_DIR and ICV_DIR are trapped to EL2.

TSEI, [13]

Trap all locally generated SEIs. The value is:

`0x0` Locally generated SEIs do not cause a trap to EL2.

TALL1, [12]

Trap all Non-secure EL1 accesses to ICC_* and ICV_* System registers for Group 1 interrupts to EL2. The possible values are:

`0x0` Non-Secure EL1 accesses to ICC_* and ICV_* registers for Group 1 interrupts proceed as normal.

`0x1` Non-secure EL1 accesses to ICC_* and ICV_* registers for Group 1 interrupts trap to EL2.

TALL0, [11]

Trap all Non-secure EL1 accesses to ICC_* and ICV_* System registers for Group 0 interrupts to EL2. The possible values are:

`0x0` Non-Secure EL1 accesses to ICC_* and ICV_* registers for Group 0 interrupts proceed as normal.

`0x1` Non-secure EL1 accesses to ICC_* and ICV_* registers for Group 0 interrupts trap to EL2.

TC, [10]

Trap all Non-secure EL1 accesses to System registers that are common to Group 0 and Group 1 to EL2. The possible values are:

`0x0` Non-secure EL1 accesses to common registers proceed as normal.

`0x1` Non-secure EL1 accesses to common registers trap to EL2.

RES0, [9:8]

Reserved, RES0.

VGrp1DIE, [7]

VM Group 1 Disabled Interrupt Enable. The possible values are:

`0x0` Maintenance interrupt disabled.

`0x1` Maintenance interrupt signaled when ICH_VMCR.VENG1 is 0.

VGrp1EIE, [6]

VM Group 1 Enabled Interrupt Enable. The possible values are:

`0x0` Maintenance interrupt disabled.

`0x1` Maintenance interrupt signaled when ICH_VMCR.VENG1 is 1.

VGrp0DIE, [5]

VM Group 0 Disabled Interrupt Enable. The possible values are:

`0x0` Maintenance interrupt disabled.

`0x1` Maintenance interrupt signaled when ICH_VMCR.VENG0 is 0.

VGrp0EIE, [4]

VM Group 0 Enabled Interrupt Enable. The possible values are:

`0x0` Maintenance interrupt disabled.

`0x1` Maintenance interrupt signaled when ICH_VMCR.VENG0 is 1.

NPIE, [3]

No Pending Interrupt Enable. The possible values are:

`0x0` Maintenance interrupt disabled.

`0x1` Maintenance interrupt signaled while the List registers contain no interrupts in the pending state.

LRENPIE, [2]

List Register Entry Not Present Interrupt Enable. The possible values are:

`0x0` Maintenance interrupt disabled.

`0x1` Maintenance interrupt is asserted while the EOIcount field is not 0.

UIE, [1]

Underflow Interrupt Enable. The possible values are:

`0x0` Maintenance interrupt disabled.

`0x1` Maintenance interrupt is asserted if none, or only one, of the List register entries is marked as a valid interrupt.

En, [0]

Enable. The possible values are:

`0x0` Virtual CPU interface operation disabled.

`0x1` Virtual CPU interface operation enabled.

Configurations

AArch32 System register ICH_HSR can be mapped to AArch64 System register ICH_HSR_EL2.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Generic Interrupt Controller Architecture Specification*.

B4.22 ICH_VMCR, Interrupt Controller Virtual Machine Control Register

ICH_VMCR enables the hypervisor to save and restore the virtual machine view of the GIC state.

Bit field descriptions

ICH_VMCR is a 32-bit register and is part of:

- The GIC system registers functional group.
- The Virtualization registers functional group.
- The GIC host interface control registers functional group.

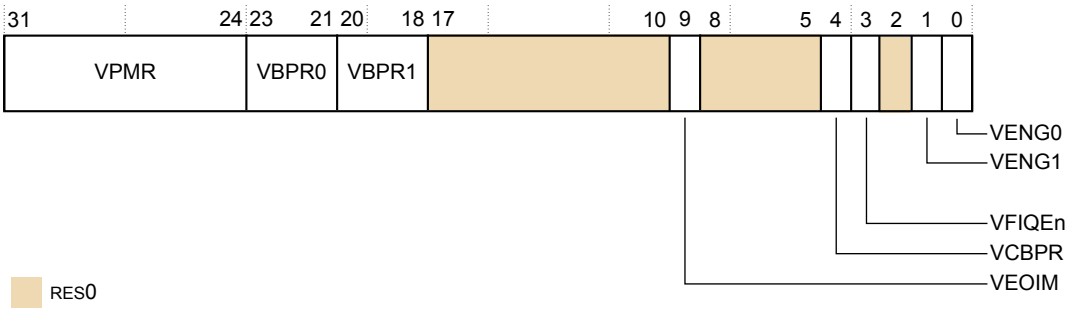

Figure B4-12 ICH_VMCR bit assignments

VPMR, [31:24]
Virtual Priority Mask.

This field is an alias of ICV_PMR.Priority.

VBPR0, [23:21]
Virtual Binary Point Register, Group 0. The minimum value is:

`0x2` This field is an alias of ICV_BPR0.BinaryPoint.

VBPR1, [20:18]
Virtual Binary Point Register, Group 1. The minimum value is:

`0x3` This field is an alias of ICV_BPR1.BinaryPoint.

[17:10]

Reserved, RES0.

VEOIM, [9]
Virtual EOI mode. The possible values are:

`0x0` ICV_EOIR0 and ICV_EOIR1 provide both priority drop and interrupt deactivation functionality. Accesses to ICV_DIR are UNPREDICTABLE.

`0x1` ICV_EOIR0 and ICV_EOIR1 provide priority drop functionality only. ICV_DIR provides interrupt deactivation functionality.

This bit is an alias of ICV_CTLR.EOImode.

[8:5]

Reserved, RES0.

VCBPR, [4]
Virtual Common Binary Point Register. The possible values are:

`0x0` ICV_BPR0 determines the preemption group for virtual Group 0 interrupts only.

ICV_BPR1 determines the preemption group for virtual Group 1 interrupts.

`0x1` ICV_BPR0 determines the preemption group for both virtual Group 0 and virtual Group 1 interrupts.

Reads of ICV_BPR1 return ICV_BPR0 plus one, saturated to 111. Writes to ICV_BPR1 are ignored.

VFIQEn, [3]

Virtual FIQ enable. The value is:

`0x1` Group 0 virtual interrupts are presented as virtual FIQs.

[2]

Reserved, RES0.

VENG1, [1]

Virtual Group 1 interrupt enable. The possible values are:

`0x0` Virtual Group 1 interrupts are disabled.

`0x1` Virtual Group 1 interrupts are enabled.

This bit is an alias of ICV_IGRPEN1.Enable.

VENG0, [0]

Virtual Group 0 interrupt enable. The possible values are:

`0x0` Virtual Group 0 interrupts are disabled.

`0x1` Virtual Group 0 interrupts are enabled.

This bit is an alias of ICV_IGRPEN0.Enable.

Configurations

AArch32 System register ICH_VMCR can be mapped to AArch64 System register ICH_VMCR_EL2.

If EL2 is not implemented, this register is RES0 from EL3.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Generic Interrupt Controller Architecture Specification*.

B4.23 ICH_VTR, Interrupt Controller VGIC Type Register

ICH_VTR reports supported GIC virtualization features.

Bit field descriptions

ICH_VTR is a 32-bit register and is part of:

- The GIC system registers functional group.
- The Virtualization registers functional group.
- The GIC host interface control registers functional group.

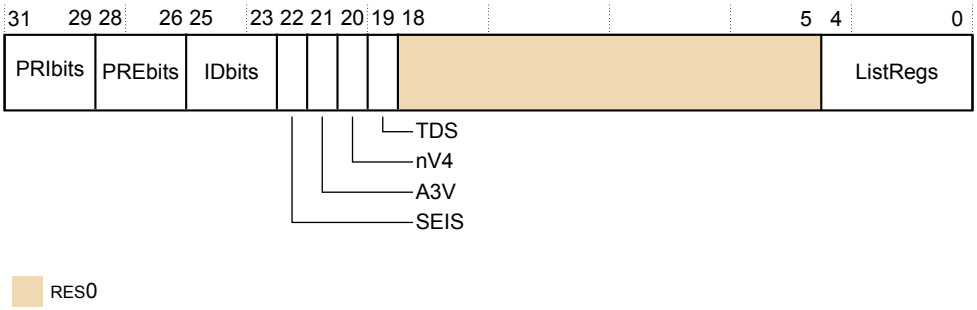

Figure B4-13 ICH_VTR bit assignments

PRIbits, [31:29]

Priority bits. The number of virtual priority bits implemented, minus one.

`0x4` Priority implemented is 5-bit.

PREbits, [28:26]

The number of virtual preemption bits implemented, minus one. The value is:

`0x4` Virtual preemption implemented is 5-bit.

IDbits, [25:23]

The number of virtual interrupt identifier bits supported. The value is:

`0x0` Virtual interrupt identifier bits implemented is 16-bit.

SEIS, [22]

SEI Support. The value is:

`0x0` The virtual CPU interface logic does not support generation of SEIs.

A3V, [21]

Affinity 3 Valid. The value is:

`0x1` The virtual CPU interface logic supports non-zero values of Affinity 3 in SGI generation System registers.

nV4, [20]

Direct injection of virtual interrupts not supported. The value is:

`0x0` The CPU interface logic supports direct injection of virtual interrupts.

TDS, [19]

Separate trapping of Non-secure EL1 writes to ICV_DIR supported. The value is:

`0x1` Implementation supports ICH_HCR.TDIR.

RES0, [18:5]

Reserved, RES0.

ListRegs, [4:0]

The number of implemented List registers, minus one. The value is:

3 The core implements four List registers.

Configurations

AArch32 System register ICH_VTR is architecturally mapped to AArch64 System register ICH_VTR_EL2.

If EL2 is not implemented, all bits in this register are RES0 from EL3, except for nV4, which is RES1 from EL3.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Generic Interrupt Controller Architecture Specification*.

B4.24 AArch64 physical GIC CPU interface system register summary

The following table lists the AArch64 physical GIC CPU interface system registers that have implementation defined bits.

See the *Arm® Generic Interrupt Controller Architecture Specification* for more information and a complete list of AArch64 physical GIC CPU interface system registers.

Table B4-4 AArch64 physical GIC CPU interface system register summary

Name	Op0	Op1	CRn	CRm	Op2	Type	Description
ICC_AP0R0_EL1	3	0	12	8	4	RW	*B4.25 ICC_AP0R0_EL1, Interrupt Controller Active Priorities Group 0 Register 0, EL1* on page B4-489
ICC_AP1R0_EL1	3	0	12	9	0	RW	*B4.26 ICC_AP1R0_EL1, Interrupt Controller Active Priorities Group 1 Register 0 EL1* on page B4-490
ICC_BPR0_EL1	3	0	12	8	3	RW	*B4.27 ICC_BPR0_EL1, Interrupt Controller Binary Point Register 0, EL1* on page B4-491
ICC_BPR1_EL1	3	0	12	12	3	RW	*B4.28 ICC_BPR1_EL1, Interrupt Controller Binary Point Register 1, EL1* on page B4-492
ICC_CTLR_EL1	3	0	12	12	4	RW	*B4.29 ICC_CTLR_EL1, Interrupt Controller Control Register, EL1* on page B4-493
ICC_CTLR_EL3	3	6	12	12	4	RW	*B4.30 ICC_CTLR_EL3, Interrupt Controller Control Register, EL3* on page B4-495
ICC_SRE_EL1	3	0	12	12	5	RW	*B4.31 ICC_SRE_EL1, Interrupt Controller System Register Enable Register, EL1* on page B4-497
ICC_SRE_EL2	3	4	12	9	5	RW	*B4.32 ICC_SRE_EL2, Interrupt Controller System Register Enable register, EL2* on page B4-498
ICC_SRE_EL3	3	6	12	12	5	RW	*B4.33 ICC_SRE_EL3, Interrupt Controller System Register Enable register, EL3* on page B4-500

B4.25 ICC_AP0R0_EL1, Interrupt Controller Active Priorities Group 0 Register 0, EL1

The ICC_AP0R0_EL1 provides information about Group 0 active priorities.

Bit descriptions

This register is a 32-bit register and is part of:

- The GIC system registers functional group.
- The GIC control registers functional group.

The core implements 5 bits of priority with 32 priority levels, corresponding to the 32 bits [31:0] of the register. The possible values for each bit are:

`0x00000000` No interrupt active. This is the reset value.

`0x00000001` Interrupt active for priority `0x0`.

`0x00000002` Interrupt active for priority `0x8`.

...

`0x80000000` Interrupt active for priority `0xF8`.

Configurations

AArch64 System register ICC_AP0R0_EL1 is architecturally mapped to AArch32 System register ICC_AP0R0.

Accessibility

The Cortex-A75 core supports 5-bit interrupt priority or 32 possible pre-emptible priorities. Accesses to ICC_AP0R0_EL1 are UNDEFINED.

Details not provided in this description are architecturally defined. See the *Arm® Generic Interrupt Controller Architecture Specification*.

B4.26 ICC_AP1R0_EL1, Interrupt Controller Active Priorities Group 1 Register 0 EL1

The ICC_AP1R0_EL1 provides information about Group 1 active priorities.

Bit descriptions

This register is a 32-bit register and is part of:

- The GIC system registers functional group.
- The GIC control registers functional group.

The core implements 5 bits of priority with 32 priority levels, corresponding to the 32 bits [31:0] of the register. The possible values for each bit are:

`0x00000000` No interrupt active. This is the reset value.

`0x00000001` Interrupt active for priority `0x0`.

`0x00000002` Interrupt active for priority `0x8`.

...

`0x80000000` Interrupt active for priority `0xF8`.

Configurations

AArch64 System register ICC_AP1R0_EL1 is architecturally mapped to AArch32 System register ICC_AP1R0.

Accessibility

The Cortex-A75 core supports 5-bit interrupt priority or 32 possible preemptable priorities. Accesses to ICC_AP1R0 are UNDEFINED.

Details not provided in this description are architecturally defined. See the *Arm® Generic Interrupt Controller Architecture Specification*.

B4.27 ICC_BPR0_EL1, Interrupt Controller Binary Point Register 0, EL1

ICC_BPR0_EL1 defines the point at which the priority value fields split into two parts, the group priority field and the subpriority field. The group priority field determines Group 0 interrupt preemption.

Bit field descriptions

ICC_BPR0_EL1 is a 32-bit register and is part of:

- The GIC system registers functional group.
- The GIC control registers functional group.

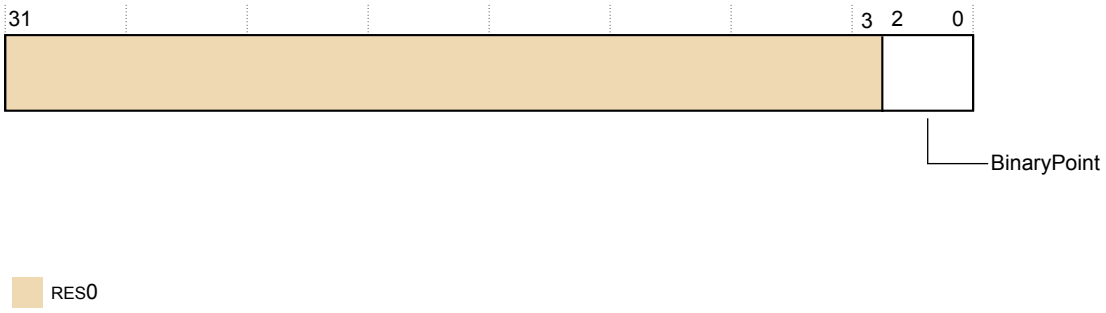

Figure B4-14 ICC_BPR0_EL1 bit assignments

RES0, [31:3]

Reserved, RES0.

BinaryPoint, [2:0]

The value of this field controls how the 8-bit interrupt priority field is split into a group priority field, that determines interrupt preemption, and a subpriority field. The minimum value that is implemented is:

`0x2`

Configurations

AArch64 System register ICC_BPR0_EL1 is architecturally mapped to AArch32 System register ICC_BPR0.

Virtual accesses to this register update ICH_VMCR_EL2.VBPR0.

Bit fields and details that are not provided in this description are architecturally defined. See the *Arm® Generic Interrupt Controller Architecture Specification*.

B4.28 ICC_BPR1_EL1, Interrupt Controller Binary Point Register 1, EL1

ICC_BPR1_EL1 defines the point at which the priority value fields split into two parts, the group priority field and the subpriority field. The group priority field determines Group 1 interrupt preemption.

Bit field descriptions

ICC_BPR1_EL1 is a 32-bit register and is part of:

- The GIC system registers functional group.
- The GIC control registers functional group.

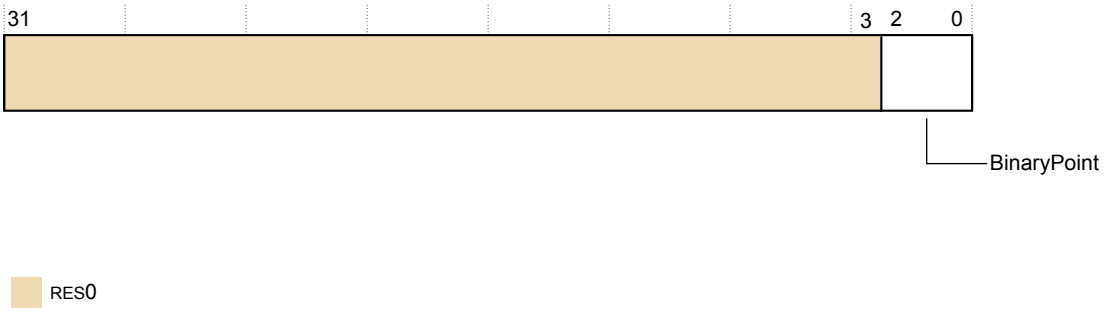

Figure B4-15 ICC_BPR1_EL1 bit assignments

RES0, [31:3]

Reserved, RES0.

BinaryPoint, [2:0]

The value of this field controls how the 8-bit interrupt priority field is split into a group priority field, that determines interrupt preemption, and a subpriority field.

The minimum value implemented of ICC_BPR1_EL1 Secure register is `0x2`.

The minimum value implemented of ICC_BPR1_EL1 Non-secure register is `0x3`.

Configurations

AArch64 System register ICC_BPR1_EL1 (S) is architecturally mapped to AArch32 System register ICC_BPR1 (S).

AArch64 System register ICC_BPR1_EL1 (NS) is architecturally mapped to AArch32 System register ICC_BPR1 (NS).

Virtual accesses to this register update ICH_VMCR_EL2.VBPR1.

Bit fields and details that are not provided in this description are architecturally defined. See the *Arm® Generic Interrupt Controller Architecture Specification*.

B4.29 ICC_CTLR_EL1, Interrupt Controller Control Register, EL1

ICC_CTLR_EL1 controls aspects of the behavior of the GIC CPU interface and provides information about the features implemented.

Bit field descriptions

ICC_CTLR_EL1 is a 32-bit register and is part of:

- The GIC system registers functional group.
- The GIC control registers functional group.

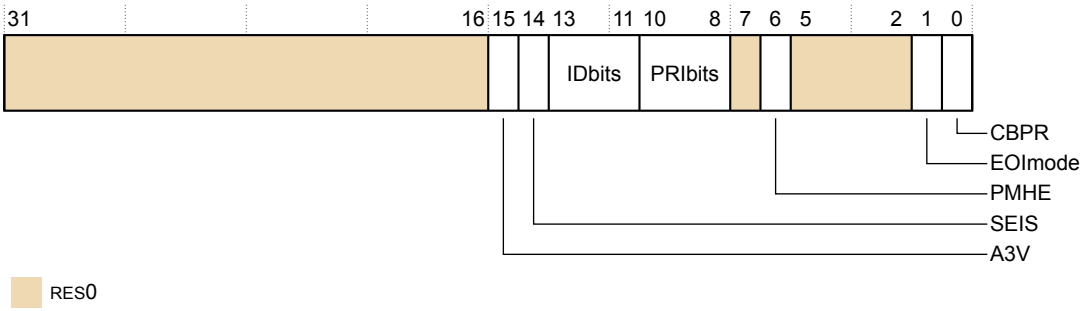

Figure B4-16 ICC_CTLR_EL1 bit assignments

RES0, [31:16]

Reserved, RES0.

A3V, [15]

Affinity 3 Valid. The value is:

`0x1` The CPU interface logic supports non-zero values of Affinity 3 in SGI generation System registers.

SEIS, [14]

SEI Support. The value is:

`0x0` The CPU interface logic does not support local generation of SEIs.

IDbits, [13:11]

Identifier bits. The value is:

`0x0` The number of physical interrupt identifier bits supported is 16 bits.

This field is an alias of ICC_CTLR_EL3.IDbits.

PRIbits, [10:8]

Priority bits. The value is:

`0x4` The core support 32 levels of physical priority with 5 priority bits.

RES0, [7]

Reserved, RES0.

PMHE, [6]

Priority Mask Hint Enable. This bit is an alias of ICC_CTLR_EL3.PMHE. The possible values are:

0 Disables use of ICC_PMR as a hint for interrupt distribution.

1 Enables use of ICC_PMR as a hint for interrupt distribution.

RES0, [5:2]

Reserved, RES0.

EOImode, [1]

End of interrupt mode for the current security state. The possible values are:

0 ICC_EOIR0 and ICC_EOIR1 provide both priority drop and interrupt deactivation functionality. Accesses to ICC_DIR are UNPREDICTABLE.

1 ICC_EOIR0 and ICC_EOIR1 provide priority drop functionality only. ICC_DIR provides interrupt deactivation functionality.

If El3 is using AArch32, this bit is an alias of ICC_MCTLR.EOImode_EL1 {S, NS}.

If EL3 is using AArch64, this bit is an alias of ICC_CTLR_EL3.EOImode_EL1 {S, NS}.

CBPR, [0]

Common Binary Point Register. Control whether the same register is used for interrupt preemption of both Group 0 and Group 1 interrupt. The possible values are:

0 ICC_BPR0 determines the preemption group for Group 0 interrupts.

ICC_BPR1 determines the preemption group for Group 1 interrupts.

1 ICC_BPR0 determines the preemption group for Group 0 and Group 1 interrupts.

If EL3 is using AArch32, this bit is an alias of ICC_MCTLR.CBPR_EL1 {N, NS}.

If EL3 is using AArch64, this bit is an alias of ICC_CTLR_EL3.CBPR_EL1 {S, NS}.

If GICD_CTLR.DS == 0, this bit is read-only.

If GICD_CTLR.DS == 0, this bit is read/write.

Configurations

AArch64 System register ICC_CTLR_EL1 (S) is architecturally mapped to AArch32 System register ICC_CTLR (S).

AArch64 System register ICC_CTLR_EL1 (NS) is architecturally mapped to AArch32 System register ICC_CTLR(NS).

Bit fields and details that are not provided in this description are architecturally defined. See the *Arm® Generic Interrupt Controller Architecture Specification*.

B4.30 ICC_CTLR_EL3, Interrupt Controller Control Register, EL3

ICC_CTLR_EL3 controls aspects of the behavior of the GIC CPU interface and provides information about the features implemented.

Bit field descriptions

ICC_CTLR_EL3 is a 32-bit register and is part of:

- The GIC system registers functional group.
- The Security registers functional group.
- The GIC control registers functional group.

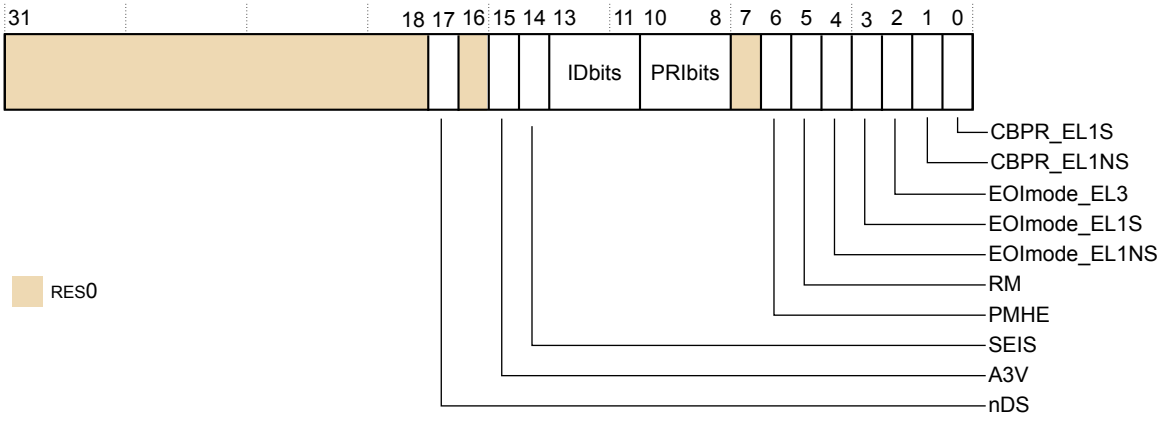

Figure B4-17 ICC_CTLR_EL3 bit assignments

RES0, [31:18]

Reserved, RES0.

nDS, [17]

Disable Security not supported. Read-only and writes are ignored. The value is:

`0x1` The CPU interface logic does not support disabling of security, and requires that security is not disabled.

RES0, [16]

Reserved, RES0.

A3V, [15]

Affinity 3 Valid. This bit is RAO/WI.

SEIS, [14]

SEI Support. The value is:

`0x0` The CPU interface logic does not support generation of SEIs.

IDbits, [13:11]

Identifier bits. The value is:

`0x0` The number of physical interrupt identifier bits supported is 16 bits.

This field is an alias of ICC_CTLR_EL3.IDbits.

PRIbits, [10:8]

Priority bits. The value is:

`0x4` The core support 32 levels of physical priority with 5 priority bits.
Accesses to ICC_AP0R{1—3} and ICC_AP1R{1—3} are UNDEFINED.

RES0, [7]

Reserved, RES0.

PMHE, [6]

Priority Mask Hint Enable. The possible values are:

0 Disables use of ICC_PMR as a hint for interrupt distribution.

1 Enables use of ICC_PMR as a hint for interrupt distribution.

RM, [5]

Routing Modifier. This bit is RAZ/WI.

EOImode_EL1NS, [4]

EOI mode for interrupts handled at Non-secure EL1 and EL2.

Controls whether a write to an End of Interrupt register also deactivates the interrupt.

EOImode_EL1S, [3]

EOI mode for interrupts handled at Secure EL1.

Controls whether a write to an End of Interrupt register also deactivates the interrupt.

EOImode_EL3, [2]

EOI mode for interrupts handled at EL3.

Controls whether a write to an End of Interrupt register also deactivates the interrupt.

CBPR_EL1NS, [1]

Common Binary Point Register, EL1 Non-secure.

Control whether the same register is used for interrupt preemption of both Group 0 and Group 1 Non-secure interrupts at EL1 and EL2.

CBPR_EL1S, [0]

Common Binary Point Register, EL1 Secure.

Control whether the same register is used for interrupt preemption of both Group 0 and Group 1 Secure interrupt at EL1.

Configurations

AArch64 System register ICC_CTLR_EL3 can be mapped to AArch32 System register ICC_MCTLR.

Bit fields and details that are not provided in this description are architecturally defined. See the *Arm® Generic Interrupt Controller Architecture Specification*.

B4.31 ICC_SRE_EL1, Interrupt Controller System Register Enable Register, EL1

ICC_SRE_EL1 controls whether the System register interface or the memory-mapped interface to the GIC CPU interface is used for EL0 and EL1.

Bit field descriptions

ICC_SRE_EL1 is a 32-bit register and is part of:

- The GIC system registers functional group.
- The GIC control registers functional group.

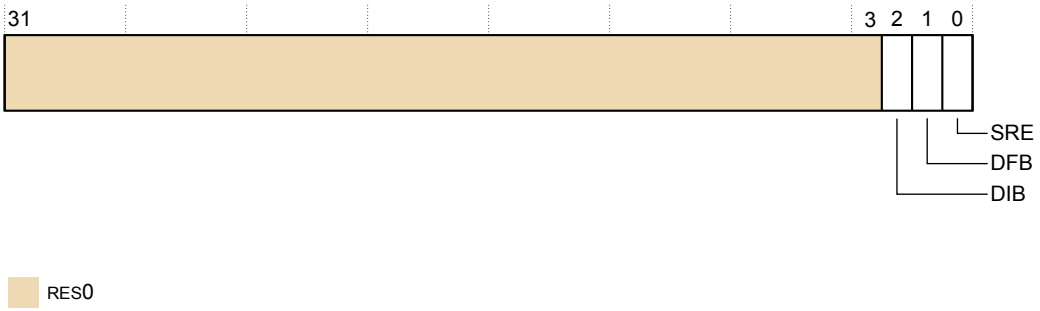

Figure B4-18 ICC_SRE bit assignments

RES0, [31:3]

Reserved, RES0.

DIB, [2]

Disable IRQ bypass. The possible values are:

`0x0` IRQ bypass enabled.

`0x1` IRQ bypass disabled.

This bit is an alias of ICC_SRE_EL3.DIB

DFB, [1]

Disable FIQ bypass. The possible values are:

`0x0` FIQ bypass enabled.

`0x1` FIQ bypass disabled.

This bit is an alias of ICC_SRE_EL3.DFB

SRE, [0]

System Register Enable. The value is:

`0x1` The System register interface for the current Security state is enabled.

This bit is RAO/WI. The core only supports a system register interface to the GIC CPU interface.

Configurations

AArch64 System register ICC_SRE_EL1 (S) is architecturally mapped to AArch32 System register ICC_SRE (S).

AArch64 System register ICC_SRE_EL1 (NS) is architecturally mapped to AArch32 System register ICC_SRE (NS).

Bit fields and details that are not provided in this description are architecturally defined. See the *Arm® Generic Interrupt Controller Architecture Specification*.

B4.32 ICC_SRE_EL2, Interrupt Controller System Register Enable register, EL2

ICC_SRE_EL2 controls whether the system register interface or the memory-mapped interface to the GIC CPU interface is used for EL2.

Bit field descriptions

ICC_SRE_EL2 is a 32-bit register and is part of:

- The GIC system registers functional group.
- The Virtualization registers functional group.
- The GIC control registers functional group.

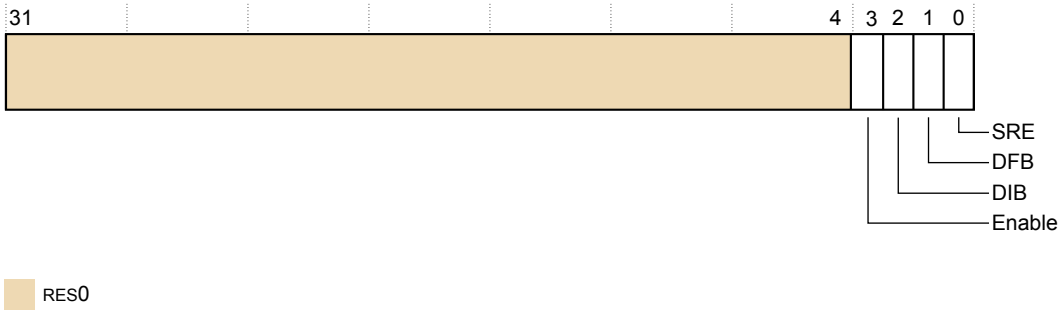

Figure B4-19 ICC_SRE_EL2 bit assignments

RES0, [31:4]

Reserved, RES0.

Enable, [3]

Enables lower Exception level access to ICC_SRE_EL1. The value is:

`0x1` Non-secure EL1 accesses to ICC_SRE_EL1 do not trap to EL2.

This bit is RAO/WI.

DIB, [2]

Disable IRQ bypass. The possible values are:

`0x0` IRQ bypass enabled.

`0x1` IRQ bypass disabled.

This bit is an alias of ICC_SRE_EL3.DIB

DFB, [1]

Disable FIQ bypass. The possible values are:

`0x0` FIQ bypass enabled.

`0x1` FIQ bypass disabled.

This bit is an alias of ICC_SRE_EL3.DFB

SRE, [0]

System Register Enable. The value is:

`0x1` The System register interface for the current Security state is enabled.

This bit is RAO/WI. The core only supports a system register interface to the GIC CPU interface.

Configurations

AArch64 System register ICC_SRE_EL2 is architecturally mapped to AArch32 System register ICC_HSRE.

If EL2 is not implemented, this register is RES0 from EL3.

Bit fields and details that are not provided in this description are architecturally defined. See the *Arm® Generic Interrupt Controller Architecture Specification*.

B4.33 ICC_SRE_EL3, Interrupt Controller System Register Enable register, EL3

ICC_SRE_EL3 controls whether the System register interface or the memory-mapped interface to the GIC CPU interface is used for EL3.

Bit field descriptions

ICC_SRE_EL3 is a 32-bit register and is part of:

- The GIC system registers functional group.
- The Security registers functional group.
- The GIC control registers functional group.

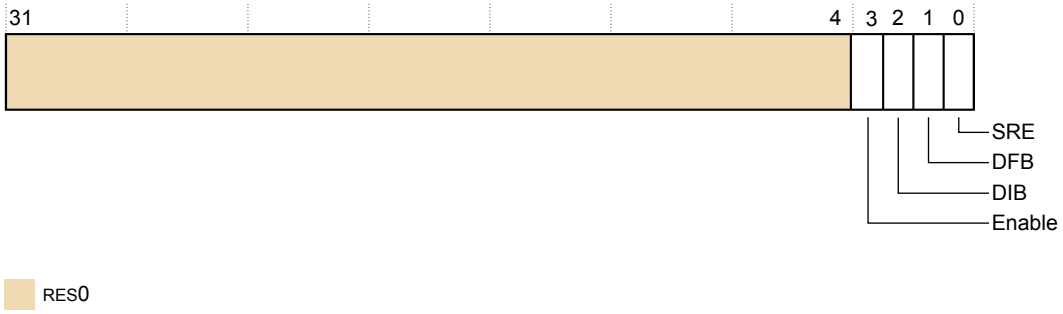

Figure B4-20 ICC_SRE_EL3 bit assignments

RES0, [31:4]

Reserved, RES0.

Enable, [3]

Enables lower Exception level access to ICC_SRE_EL1 and ICC_SRE_EL2. The value is:

1
- Secure EL1 accesses to Secure ICC_SRE_EL1 do not trap to EL3.
- EL2 accesses to Non-secure ICC_SRE_EL1 and ICC_SRE_EL2 do not trap to EL3.
- Non-secure EL1 accesses to ICC_SRE_EL1 do not trap to EL3.

This bit is RAO/WI.

DIB, [2]

Disable IRQ bypass. The possible values are:

0 IRQ bypass enabled.

1 IRQ bypass disabled.

DFB, [1]

Disable FIQ bypass. The possible values are:

0 FIQ bypass enabled.

1 FIQ bypass disabled.

SRE, [0]

System Register Enable. The value is:

1 The System register interface for the current Security state is enabled.

This bit is RAO/WI. The core only supports a system register interface to the GIC CPU interface.

Configurations

AArch64 System register ICC_SRE_EL3 can be mapped to AArch32 System register ICC_MSRE.

Bit fields and details that are not provided in this description are architecturally defined. See the *Arm® Generic Interrupt Controller Architecture Specification*.

B4.34 AArch64 virtual GIC CPU interface register summary

The following table describes the AArch64 virtual GIC CPU interface system registers that have IMPLEMENTATION DEFINED bits.

See the *Arm® Generic Interrupt Controller Architecture Specification* for more information and a complete list of AArch64 virtual GIC CPU interface system registers.

Table B4-5 AArch64 virtual GIC CPU interface register summary

Name	Op0	Op1	CRn	CRm	Op2	Type	Description
ICV_AP0R0_EL1	3	0	12	8	4	RW	*B4.35 ICV_AP0R0_EL1, Interrupt Controller Virtual Active Priorities Group 0 Register 0, EL1* on page B4-503
ICV_AP1R0_EL1	3	0	12	9	0	RW	*B4.36 ICV_AP1R0_EL1, Interrupt Controller Virtual Active Priorities Group 1 Register 0, EL1* on page B4-504
ICV_BRP0_EL1	3	0	12	8	3	RW	*B4.37 ICV_BPR0_EL1, Interrupt Controller Virtual Binary Point Register 0, EL1* on page B4-505
ICV_BPR1_EL1	3	0	12	12	3	RW	*B4.38 ICV_BPR1_EL1, Interrupt Controller Virtual Binary Point Register 1, EL1* on page B4-506
ICV_CTLR_EL1	3	0	12	12	4	RW	*B4.39 ICV_CTLR_EL1, Interrupt Controller Virtual Control Register, EL1* on page B4-507

B4.35 ICV_AP0R0_EL1, Interrupt Controller Virtual Active Priorities Group 0 Register 0, EL1

The ICV_AP0R0_EL1 register provides information about virtual Group 0 active priorities.

Bit descriptions

This register is a 32-bit register and is part of the virtual GIC system registers functional group.

The core implements 5 bits of priority with 32 priority levels, corresponding to the 32 bits [31:0] of the register. The possible values for each bit are:

`0x00000000` No interrupt active. This is the reset value.

`0x00000001` Interrupt active for priority `0x0`.

`0x00000002` Interrupt active for priority `0x8`.

...

`0x80000000` Interrupt active for priority `0xF8`.

Configurations

AArch64 System register ICV_AP0R0_EL1 is architecturally mapped to AArch32 System register ICV_AP0R0.

Details that are not provided in this description are architecturally defined. See the *Arm® Generic Interrupt Controller Architecture Specification*.

B4.36 ICV_AP1R0_EL1, Interrupt Controller Virtual Active Priorities Group 1 Register 0, EL1

The ICV_AP1R0_EL1 register provides information about virtual Group 1 active priorities.

Bit descriptions

This register is a 32-bit register and is part of the virtual GIC system registers functional group.

The core implements 5 bits of priority with 32 priority levels, corresponding to the 32 bits [31:0] of the register. The possible values for each bit are:

`0x00000000` No interrupt active. This is the reset value.

`0x00000001` Interrupt active for priority `0x0`.

`0x00000002` Interrupt active for priority `0x8`.

...

`0x80000000` Interrupt active for priority `0xF8`.

Configurations

AArch64 System register ICV_AP1R0_EL1 is architecturally mapped to AArch32 System register ICV_AP1R0.

Details that are not provided in this description are architecturally defined. See the *Arm® Generic Interrupt Controller Architecture Specification*.

B4.37 ICV_BPR0_EL1, Interrupt Controller Virtual Binary Point Register 0, EL1

ICV_BPR0_EL1 defines the point at which the priority value fields split into two parts, the group priority field and the subpriority field. The group priority field determines virtual Group 0 interrupt preemption.

Bit field descriptions

ICC_BPR0_EL1 is a 32-bit register and is part of the virtual GIC system registers functional group.

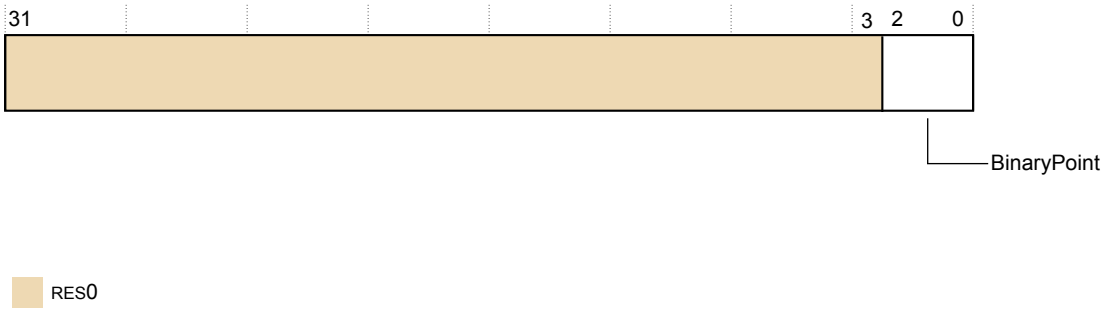

Figure B4-21 ICV_BPR0_EL1 bit assignments

RES0, [31:3]

Reserved, RES0.

BinaryPoint, [2:0]

The value of this field controls how the 8-bit interrupt priority field is split into a group priority field, that determines interrupt preemption, and a subpriority field. The minimum value that is implemented is:

`0x2`

Configurations

AArch64 System register ICV_BPR0_EL1 is architecturally mapped to AArch32 System register ICV_BPR0.

Bit fields and details that are not provided in this description are architecturally defined. See the *Arm® Generic Interrupt Controller Architecture Specification*.

B4.38 ICV_BPR1_EL1, Interrupt Controller Virtual Binary Point Register 1, EL1

ICV_BPR1_EL1 defines the point at which the priority value fields split into two parts, the group priority field and the subpriority field. The group priority field determines virtual Group 1 interrupt preemption.

Bit field descriptions

ICC_BPR1_EL1 is a 32-bit register and is part of the virtual GIC system registers functional group.

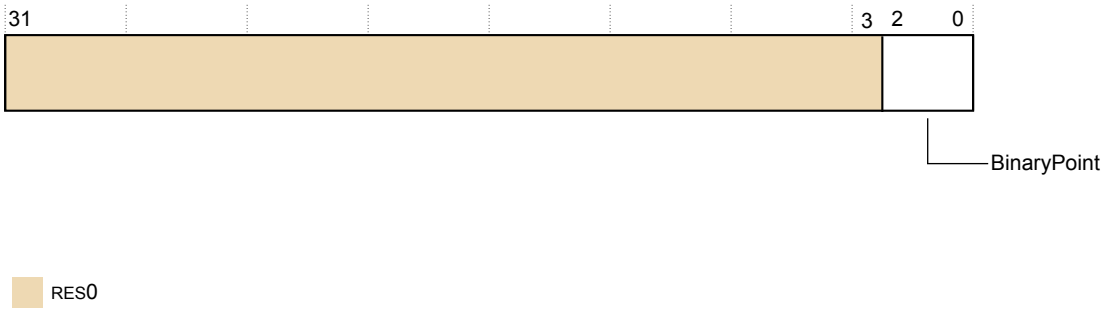

Figure B4-22 ICV_BPR1_EL1 bit assignments

RES0, [31:3]

Reserved, RES0.

BinaryPoint, [2:0]

The value of this field controls how the 8-bit interrupt priority field is split into a group priority field, that determines interrupt preemption, and a subpriority field.

The minimum value that is implemented of ICV_BPR1_EL1 Secure register is `0x2`.

The minimum value that is implemented of ICV_BPR1_EL1 Non-secure register is `0x3`.

Configurations

AArch64 System register ICV_BPR1_EL1 is architecturally mapped to AArch32 System register ICV_BPR1.

Bit fields and details that are not provided in this description are architecturally defined. See the *Arm® Generic Interrupt Controller Architecture Specification*.

B4.39 ICV_CTLR_EL1, Interrupt Controller Virtual Control Register, EL1

ICV_CTLR_EL1 controls aspects of the behavior of the GIC virtual CPU interface and provides information about the features implemented.

Bit field descriptions

ICV_CTLR_EL1 is a 32-bit register and is part of the virtual GIC system registers functional group.

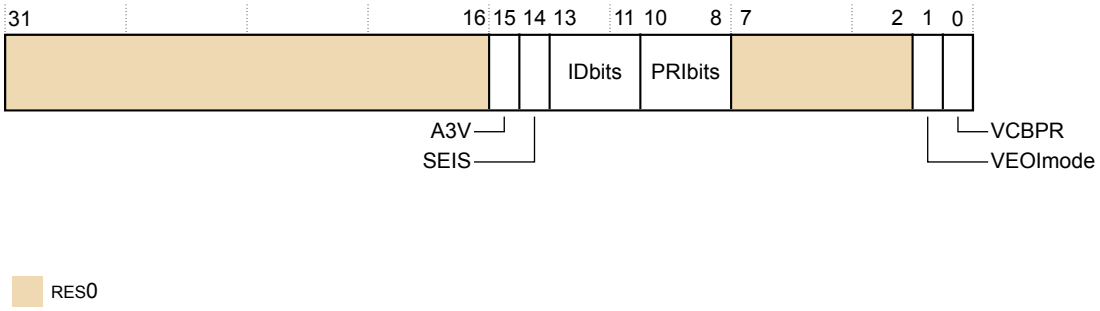

Figure B4-23 ICV_CTLR_EL1 bit assignments

RES0, [31:16]

Reserved, RES0.

A3V, [15]

Affinity 3 Valid. The value is:

`0x1` The virtual CPU interface logic supports non-zero values of Affinity 3 in SGI generation System registers.

SEIS, [14]

SEI Support. The value is:

`0x0` The virtual CPU interface logic does not support local generation of SEIs.

IDbits, [13:11]

Identifier bits. The value is:

`0x0` The number of physical interrupt identifier bits supported is 16 bits.

PRIbits, [10:8]

Priority bits. The value is:

`0x4` Support 32 levels of physical priority (5 priority bits).

RES0, [7:2]

Reserved, RES0.

VEOImode, [1]

Virtual EOI mode. The possible values are:

`0x0` ICV_EOIR0_EL1 and ICV_EOIR1_EL1 provide both priority drop and interrupt deactivation functionality. Accesses to ICV_DIR_EL1 are UNPREDICTABLE.

`0x1` ICV_EOIR0_EL1 and ICV_EOIR1_EL1 provide priority drop functionality only. ICV_DIR provides interrupt deactivation functionality.

VCBPR, [0]

Common Binary Point Register. Controls whether the same register is used for interrupt preemption of both virtual Group 0 and virtual Group 1 interrupts. The possible values are:

0 ICV_BPR0_EL1 determines the preemption group for virtual Group 0 interrupts only.

ICV_BPR1_EL1 determines the preemption group for virtual Group 1 interrupts.

1 ICV_BPR0_EL1 determines the preemption group for both virtual Group 0 and virtual Group 1 interrupts.

Reads of ICV_BPR1_EL1 return ICV_BPR0_EL1 plus one, saturated to 111. Writes to ICV_BPR1_EL1 are IGNORED.

Configurations

AArch64 System register ICV_CTLR_EL1 is architecturally mapped to AArch32 System register ICV_CTLR.

Bit fields and details that are not provided in this description are architecturally defined. See the *Arm® Generic Interrupt Controller Architecture Specification*.

B4.40 AArch64 virtual interface control system register summary

The following table lists the AArch64 virtual interface control system registers that have IMPLEMENTATION DEFINED bits.

See the *Arm® Generic Interrupt Controller Architecture Specification* for more information and a complete list of AArch64 virtual interface control system registers.

Table B4-6 AArch64 virtual interface control system register summary

Name	Op0	Op1	CRn	CRm	Op2	Type	Description
ICH_AP0R0_EL1	3	0	12	8	4	RW	*B4.41 ICH_AP0R0_EL2, Interrupt Controller Hyp Active Priorities Group 0 Register 0, EL2* on page B4-510
ICH_AP1R0_EL1	3	0	19	9	0	RW	*B4.42 ICH_AP1R0_EL2, Interrupt Controller Hyp Active Priorities Group 1 Register 0, EL2* on page B4-511
ICH_HCR_EL2	3	4	12	11	0	RW	*B4.43 ICH_HCR_EL2, Interrupt Controller Hyp Control Register, EL2* on page B4-512
ICH_VTR_EL2	3	4	12	11	1	RO	*B4.44 ICH_VMCR_EL2, Interrupt Controller Virtual Machine Control Register, EL2* on page B4-515
ICH_VMCR_EL2	3	4	12	11	7	RW	*B4.45 ICH_VTR_EL2, Interrupt Controller VGIC Type Register, EL2* on page B4-517

B4.41 ICH_AP0R0_EL2, Interrupt Controller Hyp Active Priorities Group 0 Register 0, EL2

The ICH_AP0R0 provides information about Group 0 active priorities for EL2.

Bit field descriptions

This register is a 32-bit register and is part of:

- The GIC system registers functional group.
- The Virtualization registers functional group.
- The GIC host interface control registers functional group.

The core implements 5 bits of priority with 32 priority levels, corresponding to the 32 bits [31:0] of the register. The possible values for each bit are:

`0x00000000` No interrupt active. This is the reset value.

`0x00000001` Interrupt active for priority `0x0`.

`0x00000002` Interrupt active for priority `0x8`.

...

`0x80000000` Interrupt active for priority `0xF8`.

Configurations AArch64 System register ICH_AP0R0_EL2 is architecturally mapped to AArch32 System register ICH_AP0R0.

If EL2 is not implemented, this register is RES0 from EL3.

Details that are not provided in this description are architecturally defined. See the *Arm® Generic Interrupt Controller Architecture Specification*.

B4.42 ICH_AP1R0_EL2, Interrupt Controller Hyp Active Priorities Group 1 Register 0, EL2

The ICH_AP1R0_EL2 provides information about Group 1 active priorities for EL2.

Bit field descriptions

This register is a 32-bit register and is part of:

- The GIC system registers functional group.
- The Virtualization registers functional group.
- The GIC host interface control registers functional group.

The core implements 5 bits of priority with 32 priority levels, corresponding to the 32 bits [31:0] of the register. The possible values for each bit are:

`0x00000000` No interrupt active. This is the reset value.

`0x00000001` Interrupt active for priority `0x0`.

`0x00000002` Interrupt active for priority `0x8`.

...

`0x80000000` Interrupt active for priority `0xF8`.

Configurations AArch64 System register ICH_AP1R0_EL2 is architecturally mapped to AArch32 System register ICH_AP1R0.

If EL2 is not implemented, this register is RES0 from EL3.

Details that are not provided in this description are architecturally defined. See the *Arm® Generic Interrupt Controller Architecture Specification*.

B4.43 ICH_HCR_EL2, Interrupt Controller Hyp Control Register, EL2

ICH_HCR_EL2 controls the environment for VMs.

Bit field descriptions

ICH_HCR_EL2 is a 32-bit register and is part of:

- The GIC system registers functional group.
- The Virtualization registers functional group.
- The GIC host interface control registers functional group.

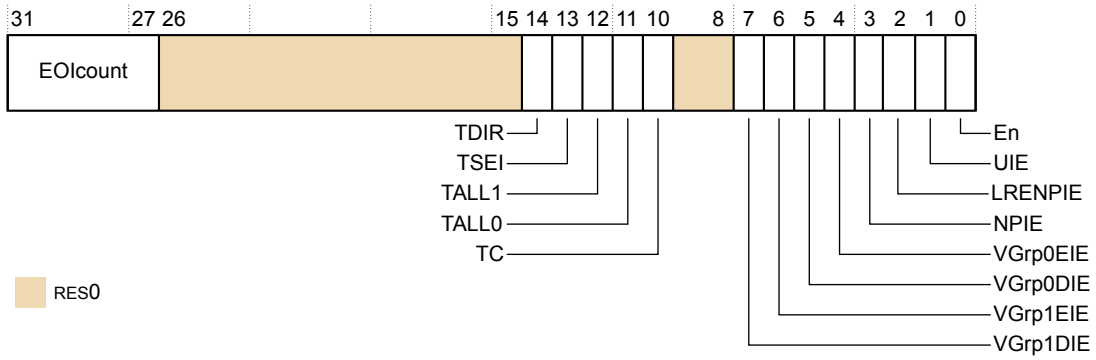

Figure B4-24 ICH_HCR_EL2 bit assignments

EOIcount, [31:27]

Number of outstanding deactivates.

RES0, [26:15]

Reserved, RES0.

TDIR, [14]

Trap Non-secure EL1 writes to ICC_DIR_EL1 and ICV_DIR_EL1. The possible values are:

`0x0` Non-secure EL1 writes of ICC_DIR_EL1 and ICV_DIR_EL1 are not trapped to EL2, unless trapped by other mechanisms.

`0x1` Non-secure EL1 writes of ICC_DIR_EL1 and ICV_DIR_EL1 are trapped to EL2.

TSEI, [13]

Trap all locally generated SEIs. The value is:

0 Locally generated SEIs do not cause a trap to EL2.

TALL1, [12]

Trap all Non-secure EL1 accesses to ICC_* and ICV_* System registers for Group 1 interrupts to EL2. The possible values are:

`0x0` Non-secure EL1 accesses to ICC_* and ICV_* registers for Group 1 interrupts proceed as normal.

`0x1` Non-secure EL1 accesses to ICC_* and ICV_* registers for Group 1 interrupts trap to EL2.

TALL0, [11]

Trap all Non-secure EL1 accesses to ICC_* and ICV_* System registers for Group 0 interrupts to EL2. The possible values are:

`0x0` Non-secure EL1 accesses to ICC_* and ICV_* registers for Group 0 interrupts proceed as normal.

`0x1` Non-secure EL1 accesses to ICC_* and ICV_* registers for Group 0 interrupts trap to EL2.

TC, [10]

Trap all Non-secure EL1 accesses to System registers that are common to Group 0 and Group 1 to EL2. The possible values are:

`0x0` Non-secure EL1 accesses to common registers proceed as normal.

`0x1` Non-secure EL1 accesses to common registers trap to EL2.

RES0, [9:8]

Reserved, RES0.

VGrp1DIE, [7]

VM Group 1 Disabled Interrupt Enable. The possible values are:

0 Maintenance interrupt disabled.

1 Maintenance interrupt signaled when ICH_VMCR_EL2.VENG1 is 0.

VGrp1EIE, [6]

VM Group 1 Enabled Interrupt Enable. The possible values are:

0 Maintenance interrupt disabled.

1 Maintenance interrupt signaled when ICH_VMCR_EL2.VENG1 is 1.

VGrp0DIE, [5]

VM Group 0 Disabled Interrupt Enable. The possible values are:

0 Maintenance interrupt disabled.

1 Maintenance interrupt signaled when ICH_VMCR_EL2.VENG0 is 0.

VGrp0EIE, [4]

VM Group 0 Enabled Interrupt Enable. The possible values are:

0 Maintenance interrupt disabled.

1 Maintenance interrupt signaled when ICH_VMCR_EL2.VENG0 is 1.

NPIE, [3]

No Pending Interrupt Enable. The possible values are:

0 Maintenance interrupt disabled.

1 Maintenance interrupt signaled while the List registers contain no interrupts in the pending state.

LRENPIE, [2]

List Register Entry Not Present Interrupt Enable. The possible values are:

0 Maintenance interrupt disabled.

1 Maintenance interrupt is asserted while the EOIcount field is not 0.

UIE, [1]

Underflow Interrupt Enable. The possible values are:

0 Maintenance interrupt disabled.

1 Maintenance interrupt is asserted if none, or only one, of the List register entries is marked as a valid interrupt.

En, [0]

Enable. The possible values are:

0 Virtual CPU interface operation disabled.

1 Virtual CPU interface operation enabled.

Configurations

AArch64 System register ICH_HCR_EL2 is architecturally mapped to AArch32 System register ICH_HCR.

If EL2 is not implemented, this register is RES0 from EL3.

Bit fields and details that are not provided in this description are architecturally defined. See the *Arm® Generic Interrupt Controller Architecture Specification*.

B4.44 ICH_VMCR_EL2, Interrupt Controller Virtual Machine Control Register, EL2

ICH_VMCR_EL2 enables the hypervisor to save and restore the virtual machine view of the GIC state.

Bit field descriptions

ICH_VMCR_EL2 is a 32-bit register and is part of:

- The GIC system registers functional group.
- The Virtualization registers functional group.
- The GIC host interface control registers functional group.

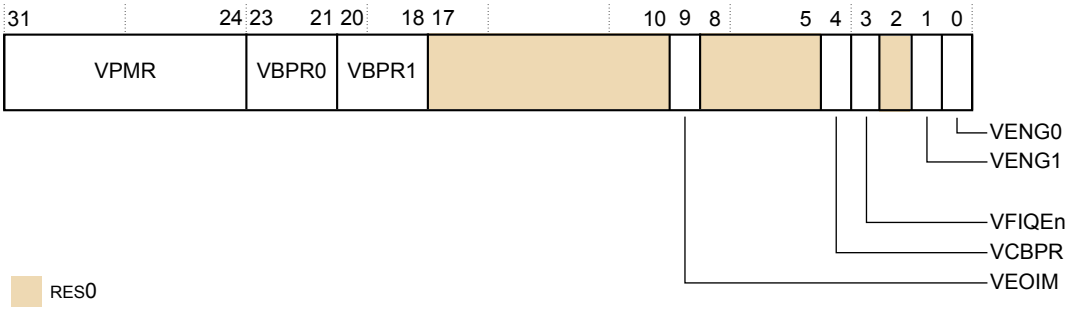

Figure B4-25 ICH_VMCR_EL2 bit assignments

VPMR, [31:24]
Virtual Priority Mask.

This field is an alias of ICV_PMR_EL1.Priority.

VBPR0, [23:21]
Virtual Binary Point Register, Group 0. The minimum value is:

`0x2` This field is an alias of ICV_BPR0_EL1.BinaryPoint.

VBPR1, [20:18]
Virtual Binary Point Register, Group 1. The minimum value is:

`0x3` This field is an alias of ICV_BPR1_EL1.BinaryPoint.

[17:10]

Reserved, RES0.

VEOIM, [9]
Virtual EOI mode. The possible values are:

`0x0` ICV_EOIR0_EL1 and ICV_EOIR1_EL1 provide both priority drop and interrupt deactivation functionality. Accesses to ICV_DIR_EL1 are UNPREDICTABLE.

`0x1` ICV_EOIR0_EL1 and ICV_EOIR1_EL1 provide priority drop functionality only. ICV_DIR_EL1 provides interrupt deactivation functionality.

This bit is an alias of ICV_CTLR_EL1.EOImode.

[8:5]

Reserved, RES0.

VCBPR, [4]
Virtual Common Binary Point Register. The possible values are:

`0x0` ICV_BPR0_EL1 determines the preemption group for virtual Group 0 interrupts only.

ICV_BPR1_EL1 determines the preemption group for virtual Group 1 interrupts.

`0x1` ICV_BPR0_EL1 determines the preemption group for both virtual Group 0 and virtual Group 1 interrupts.

Reads of ICV_BPR1_EL1 return ICV_BPR0_EL1 plus one, saturated to 111. Writes to ICV_BPR1_EL1 are IGNORED.

VFIQEn, [3]

Virtual FIQ enable. The value is:

`0x1` Group 0 virtual interrupts are presented as virtual FIQs.

[2]

Reserved, RES0.

VENG1, [1]

Virtual Group 1 interrupt enable. The possible values are:

`0x0` Virtual Group 1 interrupts are disabled.

`0x1` Virtual Group 1 interrupts are enabled.

VENG0, [0]

Virtual Group 0 interrupt enable. The possible values are:

`0x0` Virtual Group 0 interrupts are disabled.

`0x1` Virtual Group 0 interrupts are enabled.

Configurations

AArch64 System register ICH_VMCR_EL2 is architecturally mapped to AArch32 System register ICH_VMCR.

If EL2 is not implemented, this register is RES0 from EL3.

Bit fields and details that are not provided in this description are architecturally defined. See the *Arm® Generic Interrupt Controller Architecture Specification*.

B4.45 ICH_VTR_EL2, Interrupt Controller VGIC Type Register, EL2

ICH_VTR_EL2 reports supported GIC virtualization features.

Bit field descriptions

ICH_VTR_EL2 is a 32-bit register and is part of:

- The GIC system registers functional group.
- The Virtualization registers functional group.
- The GIC host interface control registers functional group.

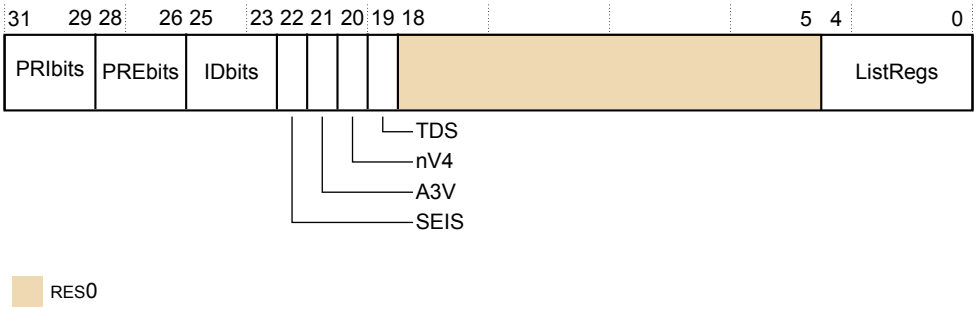

Figure B4-26 ICH_VTR_EL2 bit assignments

PRIbits, [31:29]

Priority bits. The number of virtual priority bits implemented, minus one.

`0x4` Priority implemented is 5-bit.

PREbits, [28:26]

The number of virtual preemption bits implemented, minus one. The value is:

`0x4` Virtual preemption implemented is 5-bit.

IDbits, [25:23]

The number of virtual interrupt identifier bits supported. The value is:

`0x0` Virtual interrupt identifier bits that are implemented is 16-bit.

SEIS, [22]

SEI Support. The value is:

`0x0` The virtual CPU interface logic does not support generation of SEIs.

A3V, [21]

Affinity 3 Valid. The value is:

`0x1` The virtual CPU interface logic supports non-zero values of Affinity 3 in SGI generation System registers.

nV4, [20]

Direct injection of virtual interrupts not supported. The value is:

`0x0` The CPU interface logic supports direct injection of virtual interrupts.

TDS, [19]

Separate trapping of Non-secure EL1 writes to ICV_DIR_EL1 supported. The value is:

`0x1` Implementation supports ICH_HCR_EL2.TDIR.

RES0, [18:5]

Reserved, RES0.

ListRegs, [4:0]

`0x3` The number of implemented List registers, minus one.

The core implements 4 list registers. Accesses to ICH_LR_EL2[x] (x>3) in AArch64 or ICH_LR[x]/ICH_LRC[x] (x>3) are UNDEFINED.

Configurations

AArch64 System register ICH_VTR_EL2 is architecturally mapped to AArch32 System register ICH_VTR.

If EL2 is not implemented, all bits in this register are RES0 from EL3, except for nV4, which is RES1 from EL3.

Bit fields and details that are not provided in this description are architecturally defined. See the *Arm® Generic Interrupt Controller Architecture Specification*.

Chapter B5
Advanced SIMD and floating-point registers

This chapter describes the Advanced SIMD and Floating-point registers.

It contains the following sections:

B5.1 AArch64 register summary

The core has several Advanced SIMD and floating-point system registers in the AArch64 execution state. Each register has a specific purpose, specific usage constraints, configurations, and attributes.

The following table gives a summary of the Cortex-A75 core Advanced SIMD and floating-point system registers in the AArch64 execution state.

Table B5-1 AArch64 Advanced SIMD and floating-point system registers

Name	Type	Reset	Description
FPCR	RW	0x00000000	See *B5.2.1 FPCR, Floating-point Control Register* on page B5-521.
FPSR	RW	0x00000000	See *B5.2.2 FPSR, Floating-point Status Register* on page B5-522.
MVFR0_EL1	RO	0x10110222	See *B5.2.3 MVFR0_EL1, Media and VFP Feature Register 0, EL1* on page B5-524.
MVFR1_EL1	RO	0x13211111	See *B5.2.4 MVFR1_EL1, Media and VFP Feature Register 1, EL1* on page B5-525.
MVFR2_EL1	RO	0x00000043	See *B5.2.5 MVFR2_EL1, Media and VFP Feature Register 2, EL1* on page B5-527.
FPEXC32_EL2	RW	0x00000700	See *B5.2.6 FPEXC32_EL2, Floating-point Exception Control Register, EL2* on page B5-528.

B5.2 AArch64 register descriptions

This section describes the AArch64 Advanced SIMD and floating-point system registers in the Cortex-A75 core.

This section contains the following subsections:

B5.2.1 FPCR, Floating-point Control Register

The FPCR controls floating-point behavior.

Bit field descriptions

FPCR is a 32-bit register.

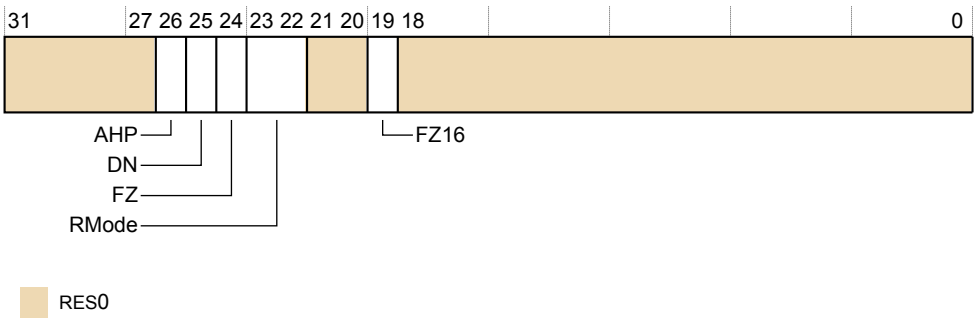

Figure B5-1 FPCR bit assignments

[31:27]

RES0 Reserved.

AHP, [26]

Alternative half-precision control bit. The possible values are:

0 IEEE half-precision format selected. This is the reset value.

1 Alternative half-precision format selected.

DN, [25]

Default NaN mode control bit. The possible values are:

0 NaN operands propagate through to the output of a floating-point operation. This is the reset value.

1 Any operation involving one or more NaNs returns the Default NaN.

FZ, [24]

Flush-to-zero mode control bit. The possible values are:

0 Flush-to-zero mode disabled. Behavior of the floating-point system is fully compliant with the IEEE 754 standard. This is the reset value.

1 Flush-to-zero mode enabled.

RMode, [23:22]

Rounding Mode control field. The encoding of this field is:

`0b00` *Round to Nearest* (RN) mode. This is the reset value.

`0b01` *Round towards Plus Infinity* (RP) mode.

`0b10` *Round towards Minus Infinity* (RM) mode.

`0b11` *Round towards Zero* (RZ) mode.

[21:20]

RES0 Reserved.

FZ16, [19]

Flush-to-zero mode control bit on half-precision data-processing instructions. The possible values are:

`0` Flush-to-zero mode disabled. Behavior of the floating-point system is fully compliant with the IEEE 754 standard. This is the default value.

`1` Flush-to-zero mode enabled.

[18:0]

RES0 Reserved.

Configurations

The named fields in this register map to the equivalent fields in the AArch32 FPSCR. See *B5.4.2 FPSCR, Floating-Point Status and Control Register* on page B5-532.

Usage constraints

Accessing the FPCR

To access the FPCR:

```
MRS <Xt>, FPCR ; Read FPCR into Xt
MSR FPCR, <Xt> ; Write Xt to FPCR
```

Register access is encoded as follows:

Table B5-2 FPCR access encoding

op0	op1	CRn	CRm	op2
11	011	0100	0100	000

Accessibility

This register is accessible as follows:

EL0	EL1 (NS)	EL1 (S)	EL2	EL3 (SCR.NS = 1)	EL3 (SCR.NS = 0)
RW	RW	RW	RW	RW	RW

B5.2.2 FPSR, Floating-point Status Register

The FPSR provides floating-point system status information.

Bit field descriptions

FPSR is a 32-bit register.

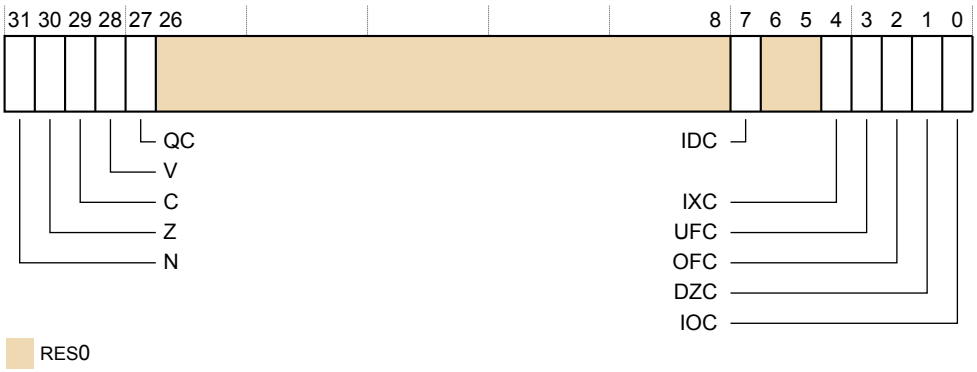

Figure B5-2 FPSR bit assignments

N, [31]
Negative condition flag for AArch32 floating-point comparison operations. AArch64 floating-point comparisons set the PSTATE.N flag instead.

Z, [30]
Zero condition flag for AArch32 floating-point comparison operations. AArch64 floating-point comparisons set the PSTATE.Z flag instead.

C, [29]
Carry condition flag for AArch32 floating-point comparison operations. AArch64 floating-point comparisons set the PSTATE.C flag instead

V, [28]
Overflow condition flag for AArch32 floating-point comparison operations. AArch64 floating-point comparisons set the PSTATE.V flag instead.

QC, [27]
Cumulative saturation bit. This bit is set to 1 to indicate that an Advanced SIMD integer operation has saturated since a 0 was last written to this bit.

[26:8]
Reserved, RES0.

IDC, [7]
Input Denormal cumulative exception bit. This bit is set to 1 to indicate that the Input Denormal exception has occurred since 0 was last written to this bit.

[6:5]
Reserved, RES0.

IXC, [4]
Inexact cumulative exception bit. This bit is set to 1 to indicate that the Inexact exception has occurred since 0 was last written to this bit.

UFC, [3]
Underflow cumulative exception bit. This bit is set to 1 to indicate that the Underflow exception has occurred since 0 was last written to this bit.

OFC, [2]
Overflow cumulative exception bit. This bit is set to 1 to indicate that the Overflow exception has occurred since 0 was last written to this bit.

DZC, [1]
Division by Zero cumulative exception bit. This bit is set to 1 to indicate that the Division by Zero exception has occurred since 0 was last written to this bit.

IOC, [0]
Invalid Operation cumulative exception bit. This bit is set to 1 to indicate that the Invalid Operation exception has occurred since 0 was last written to this bit.

Configurations
The named fields in this register map to the equivalent fields in the AArch32 FPSCR. See *B5.4.2 FPSCR, Floating-Point Status and Control Register* on page B5-532.

Usage constraints

Accessing the FPSR

To access the FPSR:

```
MRS <Xt>, FPSR; Read FPSR into Xt
MSR FPSR, <Xt>; Write Xt to FPSR
```

Register access is encoded as follows:

Table B5-3 FPSR access encoding

op0	op1	CRn	CRm	op2
11	011	0100	0100	001

Accessibility

This register is accessible as follows:

EL0	EL1 (NS)	EL1 (S)	EL2	EL3 (SCR.NS = 1)	EL3 (SCR.NS = 0)
RW	RW	RW	RW	RW	RW

B5.2.3 MVFR0_EL1, Media and VFP Feature Register 0, EL1

The MVFR0_EL1 describes the features provided by the AArch64 Advanced SIMD and floating-point implementation.

Bit field descriptions

MVFR0_EL1 is a 32-bit register.

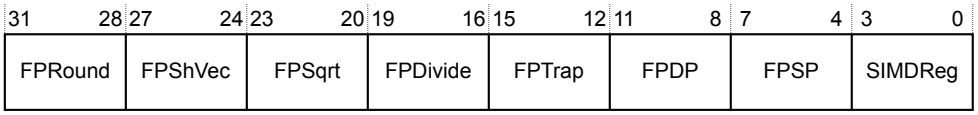

Figure B5-3 MVFR0_EL1 bit assignments

FPRound, [31:28]

Indicates the rounding modes supported by the floating-point hardware:

`1` All rounding modes supported.

FPShVec, [27:24]

Indicates the hardware support for floating-point short vectors:

`0` Not supported.

FPSqrt, [23:20]

Indicates the hardware support for floating-point square root operations:

`1` Supported.

FPDivide, [19:16]

Indicates the hardware support for floating-point divide operations:

`1` Supported.

FPTrap, [15:12]

Indicates whether the floating-point hardware implementation supports exception trapping:

`0` Not supported.

FPDP, [11:8]

Indicates the hardware support for floating-point double-precision operations:

2 Supported, VFPv3 or greater.

See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile* for more information.

FPSP, [7:4]

Indicates the hardware support for floating-point single-precision operations:

2 Supported, VFPv3 or greater.

See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile* for more information.

SIMDReg, [3:0]

Indicates support for the Advanced SIMD register bank:

2 Supported, 32 x 64-bit registers supported.

See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile* for more information.

Configurations

MVFR0_EL1 is architecturally mapped to AArch32 register MVFR0. See *B5.4.3 MVFR0, Media and VFP Feature Register 0* on page B5-535.

Usage constraints

Accessing the MVFR0_EL1

To access the MVFR0_EL1:

```
MRS <Xt>, MVFR0_EL1 ; Read MVFR0_EL1 into Xt
```

Register access is encoded as follows:

Table B5-4 MVFR0_EL1 access encoding

op0	op1	CRn	CRm	op2
11	000	0000	0011	000

Accessibility

This register is accessible as follows:

EL0	EL1(NS)	EL1(S)	EL2	EL3 (SCR.NS = 1)	EL3(SCR.NS = 0)
-	RO	RO	RO	RO	RO

B5.2.4 MVFR1_EL1, Media and VFP Feature Register 1, EL1

The MVFR1_EL1 describes the features provided by the AArch64 Advanced SIMD and floating-point implementation.

Bit field descriptions

MVFR1_EL1 is a 32-bit register.

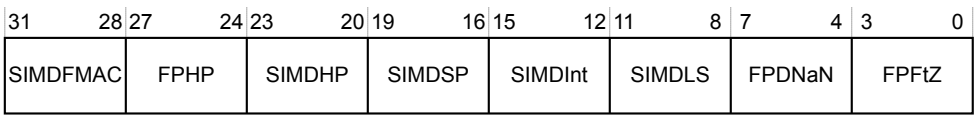

Figure B5-4 MVFR1_EL1 bit assignments

SIMDFMAC, [31:28]

Indicates whether the Advanced SIMD and floating-point unit supports fused multiply accumulate operations:

1 Implemented.

FPHP, [27:24]
Indicates whether the Advanced SIMD and floating-point unit supports half-precision floating-point conversion instructions:

3 Floating-point half precision conversion and data processing instructions implemented.

SIMDHP, [23:20]
Indicates whether the Advanced SIMD and floating-point unit supports half-precision floating-point conversion operations:

2 Advanced SIMD half precision conversion and data processing instructions implemented.

SIMDSP, [19:16]
Indicates whether the Advanced SIMD and floating-point unit supports single-precision floating-point operations:

1 Implemented.

SIMDInt, [15:12]
Indicates whether the Advanced SIMD and floating-point unit supports integer operations:

1 Implemented.

SIMDLS, [11:8]
Indicates whether the Advanced SIMD and floating-point unit supports load/store instructions:

1 Implemented.

FPDNaN, [7:4]
Indicates whether the floating-point hardware implementation supports only the Default NaN mode:

1 Hardware supports propagation of NaN values.

FPFtZ, [3:0]
Indicates whether the floating-point hardware implementation supports only the Flush-to-zero mode of operation:

1 Hardware supports full denormalized number arithmetic.

Configurations
MVFR1_EL1 is architecturally mapped to AArch32 register MVFR1. See *B5.4.4 MVFR1, Media and VFP Feature Register 1* on page B5-536.

Usage constraints

Accessing the MVFR1_EL1
To access the MVFR1_EL1:

```
MRS <Xt>, MVFR1_EL1 ; Read MVFR1_EL1 into Xt
```

Register access is encoded as follows:

Table B5-5 MVFR1_EL1 access encoding

op0	op1	CRn	CRm	op2
11	000	0000	0011	001

Accessibility

This register is accessible as follows:

EL0	EL1(NS)	EL1(S)	EL2	EL3 (SCR.NS = 1)	EL3(SCR.NS = 0)
-	RO	RO	RO	RO	RO

B5.2.5 MVFR2_EL1, Media and VFP Feature Register 2, EL1

The MVFR2_EL1 describes the features provided by the AArch64 Advanced SIMD and floating-point implementation.

Bit field descriptions

MVFR2_EL1 is a 32-bit register.

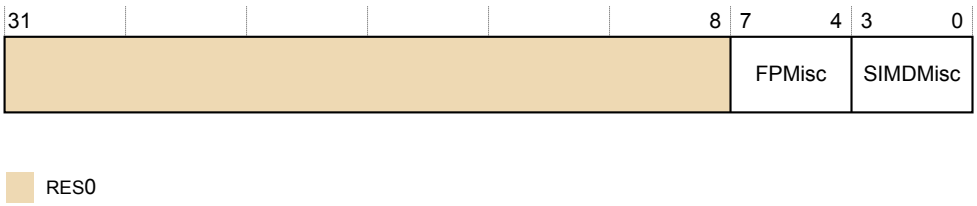

Figure B5-5 MVFR2_EL1 bit assignments

[31:8]

RES0 Reserved.

FPMisc, [7:4]

Indicates support for miscellaneous floating-point features.

`0x4` Supports:

- Floating-point selection.
- Floating-point Conversion to Integer with Directed Rounding modes.
- Floating-point Round to Integral Floating-point.
- Floating-point MaxNum and MinNum.

SIMDMisc, [3:0]

Indicates support for miscellaneous Advanced SIMD features.

`0x3` Supports:

- Floating-point Conversion to Integer with Directed Rounding modes.
- Floating-point Round to Integral Floating-point.
- Floating-point MaxNum and MinNum.

Configurations

MVFR2_EL1 is architecturally mapped to AArch32 register MVFR2. See *B5.4.5 MVFR2, Media and VFP Feature Register 2* on page B5-538.

Usage constraints

Accessing the MVFR2_EL1

To access the MVFR2_EL1:

```
MRS <Xt>, MVFR2_EL1 ; Read MVFR2_EL1 into Xt
```

Register access is encoded as follows:

Table B5-6 MVFR2_EL1 access encoding

op0	op1	CRn	CRm	op2
11	000	0000	0011	010

Accessibility

This register is accessible as follows:

EL0	EL1(NS)	EL1(S)	EL2	EL3 (SCR.NS = 1)	EL3(SCR.NS = 0)
-	RO	RO	RO	RO	RO

B5.2.6 FPEXC32_EL2, Floating-point Exception Control Register, EL2

The FPEXC32_EL2 provides access to the AArch32 register FPEXC from AArch64 state only. Its value has no effect on execution in AArch64 state.

Bit field descriptions

FPEXC32_EL2 is a 32-bit register.

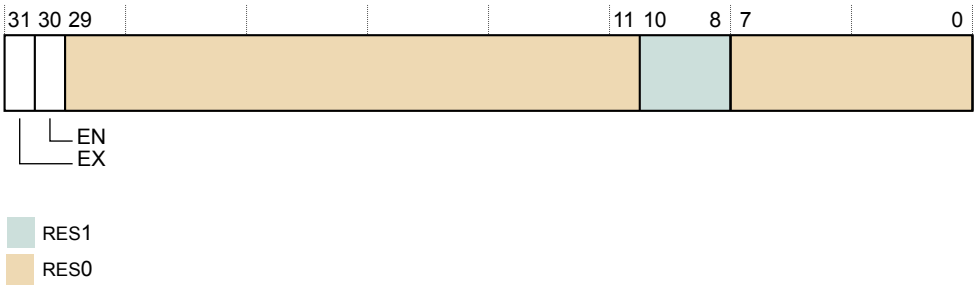

Figure B5-6 FPEXC32_EL2 bit assignments

EX, [31]

Exception bit.

RES0 The Cortex-A75 core implementation does not generate asynchronous floating-point exceptions.

EN, [30]

Enable bit. A global enable for the Advanced SIMD and floating-point support:

0 The Advanced SIMD and floating-point support is disabled. This is the reset value.

1 The Advanced SIMD and floating-point support is enabled and operates normally.

This bit applies only to AArch32 execution, and only when EL1 is not AArch64.

[29:11]

RES0 Reserved.

[10:8]

RES1 Reserved.

[7:0]

RES0 Reserved.

Configurations

FPEXC32_EL2 is architecturally mapped to AArch32 register FPEXC. See *B5.4.6 FPEXC, Floating-Point Exception Control register* on page B5-539.

Usage constraints

Accessing the FPEXC32_EL2

To access the FPEXC32_EL2:

```
MRS <Xt>, FPEXC32_EL2 ; Read FPEXC32_EL2 into Xt
             MSR FPEXC32_EL2, <Xt> ; Write Xt to FPEXC32_EL2
```

Register access is encoded as follows:

Table B5-7 FPEXC32_EL2 access encoding

op0	op1	CRn	CRm	op2
11	100	0101	0011	000

Accessibility

This register is accessible as follows:

EL0	EL1 (NS)	EL1 (S)	EL2	EL3 (SCR.NS = 1)	EL3 (SCR.NS = 0)
-	-	-	RW	RW	RW

B5.3 AArch32 register summary

The core has several Advanced SIMD and floating-point system registers in the AArch32 execution state. Each register has a specific purpose, usage constraints, configurations, and attributes.

The following table gives a summary of the Cortex-A75 core Advanced SIMD and floating-point system registers in the AArch32 execution state.

Table B5-8 AArch32 Advanced SIMD and floating-point system registers

Name	Type	Reset	Description
FPSID	RO	`0x410340A2`	See *B5.4.1 FPSID, Floating-Point System ID Register* on page B5-531.
FPSCR	RW	`0x00000000`	See *B5.4.2 FPSCR, Floating-Point Status and Control Register* on page B5-532.
MVFR0	RO	`0x10110222`	See *B5.4.3 MVFR0, Media and VFP Feature Register 0* on page B5-535.
MVFR1	RO	`0x13211111`	See *B5.4.4 MVFR1, Media and VFP Feature Register 1* on page B5-536.
MVFR2	RO	`0x00000043`	See *B5.4.5 MVFR2, Media and VFP Feature Register 2* on page B5-538.
FPEXC	RW	`0x00000700`	See *B5.4.6 FPEXC, Floating-Point Exception Control register* on page B5-539.

Note

The *Floating-Point Instruction Registers*, FPINST and FPINST2 are not implemented, and any attempt to access them is UNPREDICTABLE.

See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile* for information on permitted accesses to the Advanced SIMD and floating-point system registers.

B5.4 AArch32 register descriptions

This section describes the AArch32 Advanced SIMD and floating-point system registers in the Cortex-A75 core.

This section contains the following subsections:

B5.4.1 FPSID, Floating-Point System ID Register

The FPSID provides top-level information about the floating-point implementation.

Bit field descriptions

FPSID is a 32-bit register.

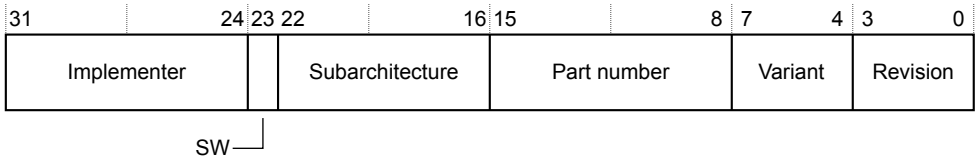

Figure B5-7 FPSID bit assignments

Implementer, [31:24]
Indicates the implementer:

`0x41` Arm Limited

SW, [23]
Software bit. This bit indicates that a system provides only software emulation of the floating-point instructions:

`0` The system includes hardware support for floating-point operations.

Subarchitecture, [22:16]
Subarchitecture version number:

`0x03` VFPv3 architecture, or later, with no subarchitecture. The entire floating-point implementation is in hardware, and requires no software support code. The MVFR0, MVFR1, and MVFR2 registers indicate the VFP architecture version.

Part number, [15:8]
Indicates the part number for the floating-point implementation:

`0x40` v8-A profile.

Variant, [7:4]
Indicates the variant number:

`A` Cortex-A75 core.

Revision, [3:0]
Indicates the revision number for the floating-point implementation:

2 r2p1.

Configurations

Access to this register depends on the values of CPACR.{cp10,cp11}, NSACR.{cp10,cp11}, and HCPTR.{TCP10,TCP11}. For details of which field values permit access at specific Exception levels, see the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

This register largely duplicates information that is held in the MIDR. Arm deprecates use of it.

Usage constraints

Accessing the FPSID

To access the FPSID:

```
VMRS <Rt>, FPSID ; Read FPSID into Rt
```

Register access is encoded as follows:

Table B5-9 FPSID access encoding

spec_reg
0000

Accessibility

This register is accessible as follows:

EL0 (NS)	EL0 (S)	EL1 (NS)	EL1 (S)	EL2	EL3 (SCR.NS = 1)	EL3 (SCR.NS = 0)
-	-	Config	RO	Config	Config	RO

B5.4.2 FPSCR, Floating-Point Status and Control Register

The FPSCR provides floating-point system status information and control.

Bit field descriptions

FPSCR is a 32-bit register.

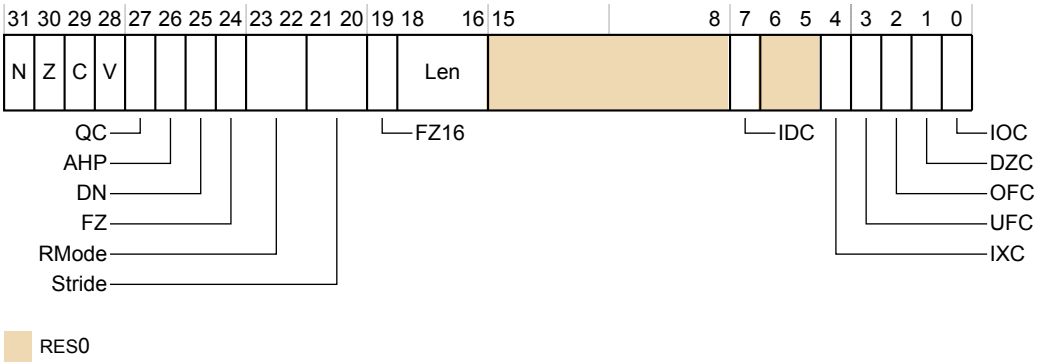

Figure B5-8 FPSCR bit assignments

N, [31]

Floating-point Negative condition code flag.

Set to 1 if a floating-point comparison operation produces a less than result.

Z, [30]

Floating-point Zero condition code flag.

Set to 1 if a floating-point comparison operation produces an equal result.

C, [29]

Floating-point Carry condition code flag.

Set to 1 if a floating-point comparison operation produces an equal, greater than, or unordered result.

V, [28]

Floating-point Overflow condition code flag.

Set to 1 if a floating-point comparison operation produces an unordered result.

QC, [27]

Cumulative saturation bit.

This bit is set to 1 to indicate that an Advanced SIMD integer operation has saturated after 0 was last written to this bit.

AHP, [26]

Alternative Half-Precision control bit:

`0` IEEE half-precision format selected. This is the reset value.

`1` Alternative half-precision format selected.

DN, [25]

Default NaN mode control bit:

`0` NaN operands propagate through to the output of a floating-point operation. This is the reset value.

`1` Any operation involving one or more NaNs returns the Default NaN.

The value of this bit only controls floating-point arithmetic. AArch32 Advanced SIMD arithmetic always uses the Default NaN setting, regardless of the value of the DN bit.

FZ, [24]

Flush-to-zero mode control bit:

`0` Flush-to-zero mode disabled. Behavior of the floating-point system is fully compliant with the IEEE 754 standard. This is the reset value.

`1` Flush-to-zero mode enabled.

The value of this bit only controls floating-point arithmetic. AArch32 Advanced SIMD arithmetic always uses the Flush-to-zero setting, regardless of the value of the FZ bit.

RMode, [23:22]

Rounding Mode control field:

`0b00` *Round to Nearest* (RN) mode. This is the reset value.

`0b01` *Round towards Plus Infinity* (RP) mode.

`0b10` *Round towards Minus Infinity* (RM) mode.

`0b11` *Round towards Zero* (RZ) mode.

The specified rounding mode is used by almost all floating-point instructions. AArch32 Advanced SIMD arithmetic always uses the Round to Nearest setting, regardless of the value of the RMode bits.

Stride, [21:20]

RES0 Reserved.

FZ16, [19]

Flush-to-zero mode control bit on half-precision data-processing instructions:

`0` Flush-to-zero mode disabled. Behavior of the floating-point system is fully compliant with the IEEE 754 standard.

`1` Flush-to-zero mode enabled.

Len, [18:16]

RES0 Reserved.

[15:8]

RES0 Reserved.

IDC, [7]

Input Denormal cumulative exception bit. This bit is set to 1 to indicate that the Input Denormal exception has occurred since 0 was last written to this bit.

[6:5]

RES0 Reserved.

IXC, [4]

Inexact cumulative exception bit. This bit is set to 1 to indicate that the Inexact exception has occurred since 0 was last written to this bit.

UFC, [3]

Underflow cumulative exception bit. This bit is set to 1 to indicate that the Underflow exception has occurred since 0 was last written to this bit.

OFC, [2]

Overflow cumulative exception bit. This bit is set to 1 to indicate that the Overflow exception has occurred since 0 was last written to this bit.

DZC, [1]

Division by Zero cumulative exception bit. This bit is set to 1 to indicate that the Division by Zero exception has occurred since 0 was last written to this bit.

IOC, [0]

Invalid Operation cumulative exception bit. This bit is set to 1 to indicate that the Invalid Operation exception has occurred since 0 was last written to this bit.

Configurations

There is one copy of this register that is used in both Secure and Non-secure states.

The named fields in this register map to the equivalent fields in the AArch64 FPCR and FPSR. See *B5.2.1 FPCR, Floating-point Control Register* on page B5-521 and *B5.2.2 FPSR, Floating-point Status Register* on page B5-522

.

Usage constraints

Accessing the FPSCR

To access the FPSCR:

```
VMRS <Rt>, FPSCR ; Read FPSCR into Rt
VMSR FPSCR, <Rt> ; Write Rt to FPSCR
```

Register access is encoded as follows:

Table B5-10 FPSCR access encoding

spec_reg
0001

Note

The Cortex-A75 core implementation does not support the deprecated VFP short vector feature. Attempts to execute the associated VFP data-processing instructions result in an UNDEFINED Instruction exception.

Accessibility

This register is accessible as follows:

EL0 (NS)	EL0 (S)	EL1 (NS)	EL1 (S)	EL2	EL3 (SCR.NS = 1)	EL3 (SCR.NS = 0)
Config	RW	Config	RW	Config	Config	RW

Access to this register depends on the values of CPACR.{cp10,cp11}, NSACR.{cp10,cp11}, HCPTR.{TCP10,TCP11} and FPEXC.EN. For details of which values of these fields allow access at which Exception levels, see the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

B5.4.3 MVFR0, Media and VFP Feature Register 0

The MVFR0 describes the features provided by the AArch32 Advanced SIMD and floating-point implementation.

Bit field descriptions

MVFR0 is a 32-bit register.

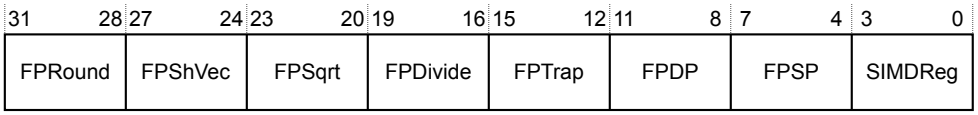

Figure B5-9 MVFR0 bit assignments

FPRound, [31:28]

Indicates the rounding modes supported by the floating-point hardware:

`0x1` All rounding modes supported.

FPShVec, [27:24]

Indicates the hardware support for floating-point short vectors:

`0x0` Not supported.

FPSqrt, [23:20]

Indicates the hardware support for floating-point square root operations:

`0x1` Supported.

FPDivide, [19:16]

Indicates the hardware support for floating-point divide operations:

`0x1` Supported.

FPTrap, [15:12]

Indicates whether the floating-point hardware implementation supports exception trapping:

`0x0` Not supported.

FPDP, [11:8]

Indicates the hardware support for floating-point double-precision operations:

`0x2` Supported, VFPv3, or greater.

See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile* for more information.

FPSP, [7:4]

Indicates the hardware support for floating-point single-precision operations:

`0x2` Supported, VFPv3, or greater.

See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile* for more information.

SIMDReg, [3:0]

Indicates support for the Advanced SIMD register bank:

`0x2` Supported, 32 x 64-bit registers supported.

See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile* for more information.

Configurations

MVFR0 is architecturally mapped to AArch64 register MVFR0_EL1. See *B5.2.3 MVFR0_EL1, Media and VFP Feature Register 0, EL1* on page B5-524.

There is one copy of this register that is used in both Secure and Non-secure states.

Usage constraints

Accessing the MVFR0

To access the MVFR0:

```
VMRS <Rt>, MVFR0 ; Read MVFR0 into Rt
```

Register access is encoded as follows:

Table B5-11 MVFR0 access encoding

spec_reg
0111

Accessibility

This register is accessible as follows:

EL0 (NS)	EL0 (S)	EL1 (NS)	EL1 (S)	EL2	EL3 (SCR.NS = 1)	EL3 (SCR.NS = 0)
-	-	Config	RO	Config	Config	RO

Access to this register depends on the values of CPACR.{cp10,cp11}, NSACR.{cp10,cp11}, HCPTR.{TCP10,TCP11}, and FPEXC.EN. For details of which values of these fields allow access at which Exception levels, see the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

MVFR0 must be interpreted with MVFR1 and MVFR2. See *B5.4.4 MVFR1, Media and VFP Feature Register 1* on page B5-536 and *B5.4.5 MVFR2, Media and VFP Feature Register 2* on page B5-538.

B5.4.4 MVFR1, Media and VFP Feature Register 1

The MVFR1 describes the features provided by the AArch32 Advanced SIMD and floating-point implementation.

Bit field descriptions

MVFR1 is a 32-bit register.

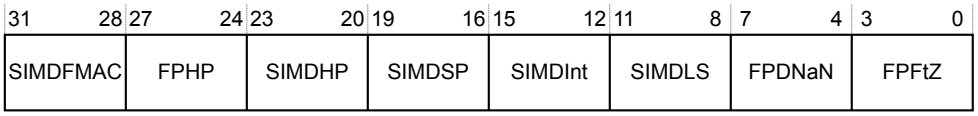

Figure B5-10 MVFR1 bit assignments

SIMDFMAC, [31:28]
Indicates whether the Advanced SIMD and floating-point unit supports fused multiply accumulate operations:

`0x1` Implemented.

FPHP, [27:24]
Indicates whether the Advanced SIMD and floating-point unit supports half-precision floating-point conversion instructions:

`0x3` Floating-point half precision conversion and data processing instructions implemented.

SIMDHP, [23:20]
Indicates whether the Advanced SIMD and floating-point unit supports half-precision floating-point conversion operations:

`0x2` Advanced SIMD precision conversion and data processing instructions implemented.

SIMDSP, [19:16]
Indicates whether the Advanced SIMD and floating-point unit supports single-precision floating-point operations:

`0x1` Implemented.

SIMDInt, [15:12]
Indicates whether the Advanced SIMD and floating-point unit supports integer operations:

`0x1` Implemented.

SIMDLS, [11:8]
Indicates whether the Advanced SIMD and floating-point unit supports load/store instructions:

`0x1` Implemented.

FPDNaN, [7:4]
Indicates whether the floating-point hardware implementation supports only the Default NaN mode:

`0x1` Hardware supports propagation of NaN values.

FPFtZ, [3:0]
Indicates whether the floating-point hardware implementation supports only the Flush-to-zero mode of operation:

`0x1` Hardware supports full denormalized number arithmetic.

Configurations
MVFR1 is architecturally mapped to AArch64 register MVFR1_EL1. See *B5.2.4 MVFR1_EL1, Media and VFP Feature Register 1, EL1* on page B5-525.

There is one copy of this register that is used in both Secure and Non-secure states.

Usage constraints

Accessing the MVFR1
To access the MVFR1:

```
VMRS <Rt>, MVFR1 ; Read MVFR1 into Rt
```

Register access is encoded as follows:

Table B5-12 MVFR1 access encoding

spec_reg
0110

Accessibility

This register is accessible as follows:

EL0 (NS)	EL0 (S)	EL1 (NS)	EL1 (S)	EL2	EL3 (SCR.NS = 1)	EL3 (SCR.NS = 0)
-	-	Config	RO	Config	Config	RO

Access to this register depends on the values of CPACR.{cp10,cp11}, NSACR.{cp10,cp11}, HCPTR.{TCP10,TCP11}, and FPEXC.EN. For details of which values of these fields allow access at which Exception levels, see the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

MVFR1 must be interpreted with MVFR0 and MVFR2. See *B5.4.3 MVFR0, Media and VFP Feature Register 0* on page B5-535 and *B5.4.5 MVFR2, Media and VFP Feature Register 2* on page B5-538.

B5.4.5 MVFR2, Media and VFP Feature Register 2

The MVFR2 describes the features provided by the AArch32 Advanced SIMD and floating-point implementation.

Bit field descriptions

MVFR2 is a 32-bit register.

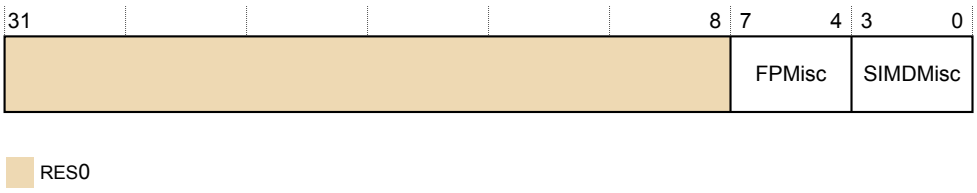

Figure B5-11 MVFR2 bit assignments

[31:8]

RES0 Reserved.

FPMisc, [7:4]

Indicates support for miscellaneous VFP features.

0x4 Supports:

- Floating-point selection.
- Floating-point Conversion to Integer with Directed Rounding modes.
- Floating-point Round to Integral Floating-point.
- Floating-point MaxNum and MinNum.

SIMDMisc, [3:0]

Indicates support for miscellaneous Advanced SIMD features.

`0x3` Supports:

- Floating-point Conversion to Integer with Directed Rounding modes.
- Floating-point Round to Integral Floating-point.
- Floating-point MaxNum and MinNum.

Configurations

MVFR2 is architecturally mapped to AArch64 register MVFR2_EL1. See *B5.2.5 MVFR2_EL1, Media and VFP Feature Register 2, EL1* on page B5-527.

There is one copy of this register that is used in both Secure and Non-secure states.

Usage constraints

Accessing the MVFR2

To access the MVFR2:

```
VMRS <Rt>, MVFR2 ; Read MVFR2 into Rt
```

Register access is encoded as follows:

Table B5-13 MVFR2 access encoding

spec_reg
0101

Accessibility

This register is accessible as follows:

EL0 (NS)	EL0 (S)	EL1 (NS)	EL1 (S)	EL2	EL3 (SCR.NS = 1)	EL3 (SCR.NS = 0)
-	-	Config	RO	Config	Config	RO

Access to this register depends on the values of CPACR.{cp10,cp11}, NSACR.{cp10,cp11}, HCPTR.{TCP10,TCP11}, and FPEXC.EN. For details of which values of these fields allow access at which Exception levels, see the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

MVFR2 must be interpreted with MVFR0 and MVFR1. See *B5.4.3 MVFR0, Media and VFP Feature Register 0* on page B5-535 and *B5.4.4 MVFR1, Media and VFP Feature Register 1* on page B5-536.

B5.4.6 FPEXC, Floating-Point Exception Control register

The FPEXC provides a global enable for the Advanced SIMD and floating-point support, and indicates how the state of this support is recorded.

Bit field descriptions

FPEXC is a 32-bit register.

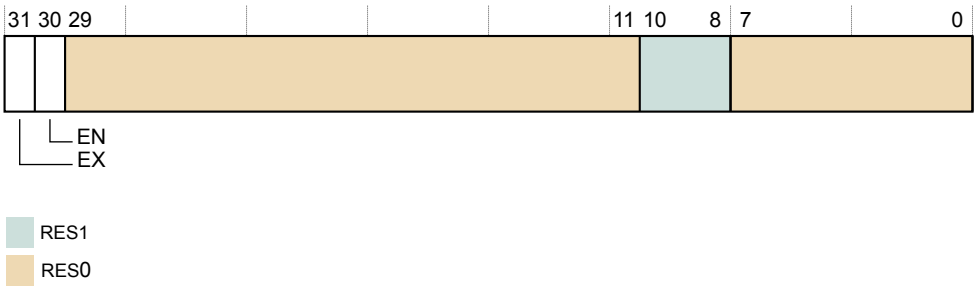

Figure B5-12 FPEXC bit assignments

EX, [31]

Exception bit. The Cortex-A75 core implementation does not generate asynchronous floating-point exceptions, therefore this bit is RES0.

EN, [30]

Global enable for the Advanced SIMD and floating-point support:

`0` The Advanced SIMD and floating-point support is disabled. This is the reset value.

`1` The Advanced SIMD and floating-point support is enabled and operates normally.

It applies only to AArch32 executions, and only when EL1 is not AArch64.

[29:11]

RES0 Reserved.

[10:8]

RES1 Reserved.

[7:0]

RES0 Reserved.

Configurations

FPEXC is architecturally mapped to AArch64 register FPEXC32_EL2. See *B5.2.6 FPEXC32_EL2, Floating-point Exception Control Register, EL2* on page B5-528.

There is one copy of this register that is used in both Secure and Non-secure states.

Usage constraints

Accessing the FPEXC

To access the FPEXC register:

```
VMRS <Rt>, FPEXC ; Read FPEXC into Rt
```

Register access is encoded as follows:

Table B5-14 FPEXC access encoding

spec_reg
1000

Accessibility

This register is accessible as follows:

EL0 (NS)	EL0 (S)	EL1 (NS)	EL1 (S)	EL2	EL3 (SCR.NS = 1)	EL3 (SCR.NS = 0)
-	-	Config	RW	Config	Config	RW

Access to this register depends on the values of CPACR.{cp10,cp11}, NSACR.{cp10,cp11}, and HCPTR.{TCP10,TCP11}. For details of which values of these fields allow access at which Exception levels, see the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

Part C
Debug descriptions

Chapter C1
Debug

This chapter describes the Cortex-A75 core debug registers and shows examples of how to use them.

It contains the following sections:

C1.1 About debug methods

The core is part of a debug system and supports both self-hosted and external debug.

The following figure shows a typical external debug system.

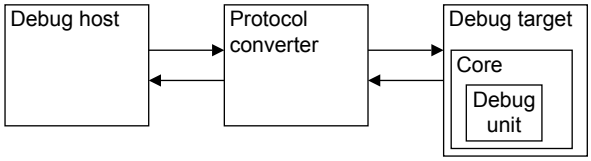

Figure C1-1 External debug system

Debug host
A computer, for example a personal computer, that is running a software debugger such as the DS-5 Debugger. With the debug host, you can issue high-level commands, such as setting a breakpoint at a certain location or examining the contents of a memory address.

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 DSTREAM is required to convert between the two protocols.

Debug target
The lowest level of the system implements system support for the protocol converter to access the debug unit using the *Advanced Peripheral Bus* (APB) slave interface. An example of a debug target is a development system with a test chip or a silicon part with a core.

Debug unit
Helps debugging software that is running on the core:
- Hardware systems that are based on the core.
- Operating systems.
- Application software.

With the debug unit, you can:
- Stop program execution.
- Examine and alter process and coprocessor state.
- Examine and alter memory and the state of the input or output peripherals.
- Restart the core.

For self-hosted debug, the debug target runs additional debug monitor software that runs on the Cortex-A75 core itself. This way, it does not require expensive interface hardware to connect a second host computer.

C1.2 Debug register interfaces

The Debug architecture defines a set of debug registers.

The debug register interfaces provide access to these registers from:

- Software running on the core.
- An external debugger.

The Cortex-A75 core implements the Armv8 Debug architecture and debug events as described in the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*. It also implements improvements to Debug introduced in v8.1 and v8.2.

C1.2.1 Core interfaces

System register access allows the core to directly access certain debug registers.

The external debug interface enables both external and self-hosted debug agents to access debug registers. Access to the debug registers is partitioned as follows:

Debug registers
This function is system register based and memory-mapped. You can access the debug register map using the APB slave port.

Performance monitor
This function is system register based and memory-mapped. You can access the performance monitor registers using the APB slave port.

Trace registers
This function is memory-mapped.

C1.2.2 Breakpoints and watchpoints

The core supports six breakpoints, four watchpoints, and a standard *Debug Communications Channel* (DCC).

A breakpoint consists of a breakpoint control register and a breakpoint value register. These two registers are referred to as a *Breakpoint Register Pair* (BRP).

Four of the breakpoints (BRP 0-3) match only to virtual address and the other two (BRP 4 and 5) match against either virtual address or context ID, or VMID. All the watchpoints can be linked to two breakpoints (BRP 4 and 5) to enable a memory request to be trapped in a given process context.

C1.2.3 Effects of resets on debug registers

The core has the following reset signals that affect the debug registers:

nCPUPORESET
This signal initializes the core logic, including the debug, ETM trace unit, breakpoint, watchpoint logic, and performance monitors logic. This maps to a Cold reset that covers reset of the core logic and the integrated debug functionality.

nCORERESET
This signal resets some of the debug and performance monitor logic. This maps to a Warm reset that covers reset of the core logic.

C1.2.4 External access permissions to debug registers

External access permission to the debug registers is subject to the conditions at the time of the access.

The following table describes the core response to accesses through the external debug interface.

Table C1-1 External access conditions to registers

Name	Condition	Description
Off	EDPRSR.PU is 0	Core power domain is completely off, or in a low-power state where the core power domain registers cannot be accessed. If debug power is off, then all external debug and memory-mapped register accesses return an error.
DLK	`DoubleLockStatus() == TRUE` (EDPRSR.DLK is 1)	OS Double Lock is locked.
OSLK	OSLSR_EL1.OSLK is 1	OS Lock is locked.
EDAD	`AllowExternalDebugAccess() ==FALSE`	External debug access is disabled. When an error is returned because of an EDAD condition code, and this is the highest priority error condition, EDPRSR.SDAD is set to 1. Otherwise SDAD is unchanged.
Default	-	None of the conditions apply, normal access.

The following table shows an example of external register access condition codes for access to a performance monitor register. To determine the access permission for the register, scan the columns from left to right. Stop at the first column a condition is true, the entry gives the access permission of the register and scanning stops.

Table C1-2 External register condition code example

Off	DLK	OSLK	EDAD	Default
-	-	-	-	RO

C1.3 Debug events

A debug event can be a software debug event or a halting debug event.

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

- Ignores the debug event.
- Takes a debug exception.
- Enters debug state.

See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile* for more information on debug events.

C1.3.1 Watchpoint debug events

In the Cortex-A75 core, watchpoint debug events are always synchronous.

Memory hint instructions and cache clean operations, except `DC ZVA`, `DC IVAC`, and `DCIMVAC`, do not generate watchpoint debug events. Store exclusive instructions generate a watchpoint debug event even when the check for the control of exclusive monitor fails. Atomic CAS instructions generate a watchpoint debug event even when the compare operation fails.

For watchpoint debug events, except those resulting from cache maintenance operations, the value reported in DFAR is guaranteed to be no lower than the address of the watchpoint location rounded down to a multiple of 16 bytes.

C1.3.2 Debug OS Lock

Debug OS Lock is set by the powerup reset, **nCPUPORESET**.

For normal behavior of debug events and debug register accesses, Debug OS Lock must be cleared. For more information, see the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

C1.4 External debug interface

For information about external debug interface, including debug memory map and debug signals, see the *Arm® DynamIQ™ Shared Unit Technical Reference Manual*.

Chapter C2
Performance Monitor Unit

This chapter describes the *Performance Monitor Unit* (PMU) and the registers that it uses.

It contains the following sections:

C2.1 About the PMU

The Cortex-A75 core includes performance monitors that enable you to gather various statistics on the operation of the core and its memory system during runtime. These provide useful information about the behavior of the core that you can use when debugging or profiling code.

The PMU provides six counters. Each counter can count any of the events available in the core. The absolute counts recorded might vary because of pipeline effects. This has negligible effect except in cases where the counters are enabled for a very short time.

C2.2 PMU functional description

This section describes the functionality of the PMU.

The PMU includes the following interfaces and counters:

Event interface
Events from all other units from across the design are provided to the PMU.

System register and APB interface
You can program the PMU registers using the system registers or the external APB interface.

Counters
The PMU has 32-bit counters that increment when they are enabled, based on events, and a 64-bit cycle counter.

PMU register interfaces
The Cortex-A75 core supports access to the performance monitor registers from the internal system register interface and a memory-mapped interface.

C2.2.1 External register access permissions

Whether or not access is permitted to a register depends on:

- If the core is powered up.
- The state of the OS Lock and OS Double Lock.
- The state of the debug authentication inputs to the core.

The behavior is specific to each register and is not described in this document. For a detailed description of these features and their effects on the registers, see the *Arm® Architecture Reference Manual Arm®v8, for Arm®v8-A architecture profile*.

The register descriptions provided in this manual describe whether each register is read/write or read-only.

C2.3 PMU events

The following table shows the events that are generated and the numbers that the PMU uses to reference the events. The table also shows the bit position of each event on the event bus. Event reference numbers that are not listed are reserved.

Table C2-1 PMU events

PMU event number	Event mnemonic	PMU event bus (to trace)	Event name	Event description
0x00	SW_INCR	-	Software increment	Instruction architecturally executed (condition check pass).
0x01	L1I_CACHE_REFILL	[0]	L1 instruction cache refill	This event counts cacheable linefill requests.
0x02	L1I_TLB_REFILL	[1]	L1 instruction TLB refill	This event counts refills from the main TLB. Refills that do not result in actual allocations in the instruction micro TLB, including translation faults and *Compare And Swap* (CAS) fails, are excluded.
0x03	L1D_CACHE_REFILL	[2]	L1 data cache refill	This event counts all allocations into the L1 cache. This includes read linefills, store linefills, and prefetch linefills.
0x04	L1D_CACHE	[4:3]	L1 data cache access	This event counts read, write and prefetch accesses to the L1 data cache. This includes non-cacheable speculative reads which do not have cacheability attributes yet. *Cache Maintenance Operation* (CMO) accesses are excluded.
0x05	L1D_TLB_REFILL	[5]	L1 data TLB refill	This event counts all refills effectively allocated in the data micro TLB. Translation faults are not counted and this even counts without taking into account whether the MMU is enabled or not.
0x08	INST_RETIRED	[13:10]	Instruction architecturally executed	This event increments for every architecturally executed instruction, including instructions that fail their condition code check.[17]
0x09	EXC_TAKEN		Exception taken	This event is set every time that an exception is executed. CCFAIL exceptions are excluded.
0x0A	EXC_RETURN	[17[18]	Instruction architecturally executed, condition code check pass, exception return	This event is set every time that an exception return is executed in ALU0. CCFAIL exceptions are excluded.
0x0B	CID_WRITE_RETIRED	[19]	Instruction architecturally executed, condition code check pass, write to CONTEXTIDR	-

Table C2-1 PMU events (continued)

PMU event number	Event mnemonic	PMU event bus (to trace)	Event name	Event description
0x0C	PC_WRITE_RETIRED	[21]	Instruction architecturally executed, condition check pass, software change of the PC	This event counts all branches taken. This excludes exception entries and debug entries.
0x0D	BR_IMMED_RETIRED	[22]	Instruction architecturally executed, immediate branch	This event counts all branches decoded as immediate branches, taken or not taken. This excludes exception entries, debug entries, and CCFAIL branches.
0x0E	BR_RETURN_RETIRED	[23]	Instruction architecturally executed, condition code check pass, procedure return	This event counts the following branches: • "BX R14" • "MOV PC, LR" • "POP{…,PC}" • "LDR PC, SP, #offset"
0x10	BR_MIS_PRED	[26]	Mispredicted or not predicted branch speculatively executed	This event counts branches mispredicted or not predicted. It counts: • Each correction to the predicted program flow that occurs because of a misprediction. • Each correction to the predicted program flow that occurs because there is no prediction. • Each correction that relates to instructions that the program flow prediction resources can predict.
0x11	CPU_CYCLES	-	Cycle	-
0x12	BR_PRED	[27]	Predictable branch speculatively executed	This event counts all updates to the program counter, apart from exception call/return.
0x13	MEM_ACCESS	-	Data memory access	-
0x14	L1I_CACHE	-	L1 instruction cache access	This event is incremented when an instruction fetch reads data from data RAM or buffer, and did not need a linefill. Cache maintenance operations are excluded. The fetch granularity is 128 bits.
0x15	L1D_CACHE_WB	-	L1 data cache Write-Back	This event counts evictions caused by natural allocation, CMO, and snoops. Write streams are excluded.
0x16	L2D_CACHE	-	L2 data cache access	This event counts data reads, instruction reads, and prefetches that hit. Snoops are not counted.
0x17	L2D_CACHE_REFILL	-	L2 data cache refill	This event counts reads and prefetches that allocate a new linefill buffer entry. The reads and prefetches that fold into a prefetch are not counted.

Table C2-1 PMU events (continued)

PMU event number	Event mnemonic	PMU event bus (to trace)	Event name	Event description
0x18	L2D_CACHE_WB	-	L2 data cache Write-Back	This event counts evictions with data caused by L2 CMOs, L2 evictions, and L2 snoops. It does not count L1 snooped data, streams, and data evicted by an L1 instruction write CMO.
0x19	BUS_ACCESS	-	Bus access	This event counts any move of data received from or sent to the SCU.
0x1B	INT_SPEC	-	Operation speculatively executed	This event counts all the instructions that go through the core rename block at each cycle. This includes instructions architecturally executed and the last micro-operation of each instruction that is not architecturally executed.
0x1C	TTBR_WRITE_RETIRED	[20]	Instruction architecturally executed (condition check pass) - Write to TTBR	-
0x1D	BUS_CYCLES	[20]	Bus cycles	-
0x1E	CHAIN	-	Chain	For odd-numbered counters, increments the count by one for each overflow of the preceding even-numbered counter. For even-numbered counters, there is no increment.
0x1F	L1D_CACHE_ALLOCATE	-	Level 1 data cache allocation without refill	-
0x20	L2D_CACHE_ALLOCATE	-	Level 2 data cache allocation without refill	-
0x21	BR_RETIRED	-	Instruction architecturally executed, branch	This event counts all branches, taken or not taken. This excludes exception entries and debug entries. In the core, an ISB is a branch and is also counted.
0x23	STALL_FRONTEND	[16]	No operation issued because of the front end	The counter counts every cycle counted by the CPU_CYCLES event on which no operation was issued because there are no operations coming from the instruction side available to issue for this core. Flush windows are excluded.
0x24	STALL_BACKEND	[15]	No operation issued because of the back end	The counter counts every cycle counted by the CPU_CYCLES event on which no operation was issued because the rename stage cannot accept any instructions coming from the decoder stage. Flush windows are excluded.

Table C2-1 PMU events (continued)

PMU event number	Event mnemonic	PMU event bus (to trace)	Event name	Event description
0x25	L1D_TLB	-	Level 1 data TLB access	This event counts all accesses to the data micro TLB, that is load and store instructions, and speculative load and store instructions executed. This event counts without taking into account whether the MMU is enabled or not. If 2 accesses are performed at the same time, then the counter is incremented twice.
0x26	L1I_TLB	-	Level 1 instruction TLB access	This event counts any instruction fetch that accesses the instruction micro TLB. This event does not take into account whether the MMU is enabled or not. The fetch granularity is 128 bits.
0x29	L3D_CACHE_ALLOCATE	-	Attributable Level 3 data or unified cache allocation without refill	-
0x2A	L3D_CACHE_REFILL	-	Attributable Level 3 data or unified cache refill	This event counts all cacheable read transactions that return from the DSU and come from outside of the cluster.
0x2B	L3D_CACHE	-	Attributable Level 3 data or unified cache access	This event counts all cacheable read transactions that return data from the SCU, and all cache line writes into the L3 cache that do not cause a linefill.
0x2D	L2D_TLB_REFILL	-	Attributable Level 2 data or unified TLB refill	This event counts refills for VA to PA translations only. The counter is not incremented on partial translations allocated in the main TLB and not incremented for IPA to VA translations. The core implements a unified TLB, therefore only L2_TLB_REFILL is incremented. This event counts only if stage 1 MMU is enabled.
0x2F	L2D_TLB	-	Attributable Level 2 data or unified TLB access	This event counts accesses to the main TLB caused by a correct requester and only if stage 1 MMU is enabled. The counter is incremented one time for each translation.
0x30	L2I_TLB	-	Attributable Level 2 instruction TLB access	This event counts accesses to the main TLB caused by a correct requester and only if stage 1 MMU is enabled. The counter is incremented one time for each translation.
0x34	DTLB_WALK	-	Data access to unified TLB that caused a page table walk	This event counts every access caused by each requester that does not hit in a VA to PA translation. The counter increments even if the translation generates a fault and only if stage 1 MMU is enabled.

Table C2-1 PMU events (continued)

PMU event number	Event mnemonic	PMU event bus (to trace)	Event name	Event description
0x35	ITLB_WALK	-	Instruction access to unified TLB that caused a page table walk	This event counts every access caused by each requester that does not hit in a VA to PA translation. The counter increments even if the translation generates a fault and only if stage 1 MMU is enabled.
0x36	LL_CACHE_RD	-	Last level cache access, read	-
0x37	LL_CACHE_MISS_RD	-	Last level cache miss, read	-
0x38	REMOTE_ACCESS_RD	-	Access to another socket in a multi-socket system, read	-
0x40	L1D_CACHE_RD	-	L1 data cache access, read	This event counts all read accesses, PLD, PF, TLB, and multiple cache accesses for the same load.
0x41	L1D_CACHE_WR	-	L1 data cache access, write	This event counts all store lookups, reads, and writes into the cache.
0x46	L1D_CACHE_WB_VICTIM	-	Level 1 data cache write-back, victim	-
0x47	L1D_CACHE_WB_CLEAN	-	Level 1 data cache write-back, cleaning and, coherency	-
0x48	L1D_CACHE_INVAL	-	Level 1 data cache invalidate	-
0x50	L2D_CACHE_RD	-	L2 data cache access, read	This event counts reads and prefetches. Snoop reads are excluded.
0x51	L2D_CACHE_WR	-	L2 data cache access, write	This event counts L1 natural evictions, cache maintenance operations evictions, snoop evictions, and streams.
0x56	L2D_CACHE_WB_VICTIM	-	L2 data cache write-back, victim	-
0x57	L2D_CACHE_WB_CLEAN	-	L2 data cache write-back, cleaning and coherency	-
0x58	L2D_CACHE_INVAL	-	L2 data cache invalidate	-
0x60	BUS_ACCESS_RD	-	Bus access read	This event counts beats of read data received from the SCU.
0x61	BUS_ACCESS_WR	-	Bus access write	This event counts beats of store data sent to the SCU.
0x66	MEM_ACCESS_RD	-	Data memory access, read	-
0x67	MEM_ACCESS_WR	-	Data memory access, write	-
0x6A	UNALIGNED_LDST_SPEC	[25:24]	Unaligned access	-

Table C2-1 PMU events (continued)

PMU event number	Event mnemonic	PMU event bus (to trace)	Event name	Event description
0x6C	LDREX_SPEC	-	Exclusive operation speculatively executed, LDREX, or LDX	-
0x6D	STREX_PASS_SPEC	-	Exclusive operation speculatively executed, STREX, or STX pass	-
0x6E	STREX_FAIL_SPEC	-	Exclusive operation speculatively executed, STREX, or STX fail	-
0x6F	STREX_SPEC	-	Exclusive operation speculatively executed, STREX, or STX	-
0x70	LD_SPEC	[7:6]	Operation speculatively executed, load	This event counts all the instructions that go through the core rename block at each cycle, similar to INST_SPEC. The counter counts the last micro-operation of each LDR instruction.
0x71	ST_SPEC	[9:8]	Operation speculatively executed, store	This event counts all the instructions that go through the core rename block at each cycle, similar to INST_SPEC. The counter counts the last micro-operation of each STR instruction.
0x72	LDST_SPEC	-	Operation speculatively executed, load or store	This event counts all the instructions that go through the core rename block at each cycle, similar to INST_SPEC. The counter counts the last micro-operation of each LDR or STR instruction.
0x73	DP_SPEC	-	Operation speculatively executed, integer data processing	This event counts all the instructions that go through the core rename block at each cycle, similar to INST_SPEC. The counter counts the last micro-operation of each data processing instruction.
0x74	ASE_SPEC	-	Operation speculatively executed, Advanced SIMD instruction	This event counts all the instructions that go through the core rename block at each cycle, similar to INST_SPEC. The counter counts the last micro-operation of each data engine SIMD instruction.
0x75	VFP_SPEC	-	Operation speculatively executed, floating-point instruction.	This event counts all the instructions that go through the core rename block at each cycle, similar to INST_SPEC. The counter counts the last micro-operation of each data engine floating-point instruction.

Table C2-1 PMU events (continued)

PMU event number	Event mnemonic	PMU event bus (to trace)	Event name	Event description
0x77	CRYPTO_SPEC	-	Operation speculatively executed, Cryptographic instruction	This event counts all the instructions that go through the core rename block at each cycle, similar to INST_SPEC. The counter counts the last micro-operation of each data engine Cryptographic instruction.
0x7A	BR_INDIRECT_SPEC	-	Branch speculatively executed - Indirect branch	-
0x7C	ISB_SPEC	-	Barrier speculatively executed, ISB	This event counts all architectural ISB instructions.
0x7D	DSB_SPEC	-	Barrier speculatively executed, DSB	-
0x7E	DMB_SPEC	-	Barrier speculatively executed, DMB	-
0x81	EXC_UNDEF	-	Counts the number of UNDEFINED exceptions taken	This event is set every time that an UNDEF exception is executed. CCFAIL exceptions are excluded.
0x8A	EXC_HVC	-	Exception taken, Hypervisor Call	This event is set every time that an exception is executed because of an HVC instruction. CCFAIL exceptions are excluded. This event is not counted when it is accessible from Non-secure EL0 or EL1.
0xA0	L3D_CACHE_RD	-	Attributable Level 3 data or unified cache access, read	This event counts `RDUnique`, `RDClean`, `RDNotSharedDirty`, atomics (`STR`, `LD`, `SWP`, `CMP` reads and writes), and `StashOnce` sent to L3.
0xA2	L3D_CACHE_REFILL_RD	-	Attributable Level 3 data or unified cache refill, read	-
0xC0	LF_STALL	-	A linefill caused an instruction side stall	-
0xC1	PTW_STALL	-	A translation table walk caused an instruction side stall	-
0xC2	I_TAG_RAM_RD	-	Number of ways read in the instruction cache - Tag RAM	-
0xC3	I_DATA_RAM_RD	-	Number of ways read in the instruction cache - Data RAM	-
0xC4	I_BTAC_RAM_RD	-	Number of ways read in the instruction - BTAC RAM	-
0xD3	D_LSU_SLOT_FULL	-	Duration for which all slots in the *Load-Store Unit* (LSU) are busy	-

Table C2-1 PMU events (continued)

PMU event number	Event mnemonic	PMU event bus (to trace)	Event name	Event description
0xD8	LS_IQ_FULL	-	Duration for which all slots in the load-store issue queue are busy	This event counts the cycles where all slots in the LS IQs are full with micro-operations waiting for issuing, and the dispatch stage is not empty.
0xD9	DP_IQ_FULL	-	Duration for which all slots in the data processing issue queue are busy	This event counts the cycles where all slots in the DP0 and DP1 IQs are full with micro-operations waiting for issuing, and the despatch stage is not empty.
0xDA	DE_IQ_FULL	-	Duration for which all slots in the data engine issue queue are busy	This event is set every time that the data engine rename has at least one valid instruction, excluding *No Operations* (NOPs), that cannot move to the issue stage because accpt_instr is LOW.
0xDC	EXC_TRAP_HYP	-	Number of traps to hypervisor	This event counts the number of exception traps taken to EL2, excluding HVC instructions. This event is set every time that an exception is executed because of a decoded trap to the hypervisor. CCFAIL exceptions and traps caused by HVC instructions are excluded. This event is not counted when it is accessible from Non-secure EL0 or EL1.
0xDE	ETM_EXT_OUT0	-	ETM trace unit output 0	-
0xDF	ETM_EXT_OUT1	-	ETM trace unit output 1	-
0xE0	MMU_PTW	-	Duration of a translation table walk handled by the MMU	-
0xE1	MMU_PTW_ST1	-	Duration of a Stage 1 translation table walk handled by the MMU	This event is not counted when it is accessible from Non-secure EL0 or EL1.
0xE2	MMU_PTW_ST2	-	Duration of a Stage 2 translation table walk handled by the MMU	This event is not counted when it is accessible from Non-secure EL0 or EL1.
0xE3	MMU_PTW_LSU	-	Duration of a translation table walk requested by the LSU	-
0xE4	MMU_PTW_ISIDE	-	Duration of a translation table walk requested by the instruction side	-
0xE5	MMU_PTW_PLD	-	Duration of a translation table walk requested by a Preload instruction or Prefetch request	-

Table C2-1 PMU events (continued)

PMU event number	Event mnemonic	PMU event bus (to trace)	Event name	Event description
0xE6	MMU_PTW_CP15	-	Duration of a translation table walk requested by an address translation operation	-
0xE7	L1PLD_TLB_REFILL	-	Level 1 PLD TLB refill	-
0xE8	L2PLD_TLB	-	Level 2 preload and MMU prefetcher TLB access	This event only counts software and hardware prefetches at Level 2.
0xE9	UTLB_FLUSH	-	Level 1 TLB flush	-
0xEA	TLB_ACCESS	-	Level 2 TLB access	-
0xEB	L1PLD_TLB	-	Level 1 preload TLB access	This event only counts software and hardware prefetches at Level 1. This event counts all accesses to the preload data micro TLB, that is L1 prefetcher and preload instructions. This event does not take into account whether the MMU is enabled or not.
0xEC	PLDTLB_WALK	-	Prefetch access to unified TLB that caused a page table walk	This event counts software and hardware prefetches.

C2.4 PMU interrupts

The Cortex-A75 core asserts the **nPMUIRQ** signal when the PMU generates an interrupt.

You can route this signal to an external interrupt controller for prioritization and masking. This is the only mechanism that signals this interrupt to the core.

This interrupt is also driven as a trigger input to the CTI. See the *Arm® DynamIQ™ Shared Unit Technical Reference Manual* for more information.

C2.5 Exporting PMU events

Some of the PMU events are exported to the ETM trace unit to be monitored.

——— **Note** ———

The **PMUEVENT** bus is not exported to external components. This is because the event bus cannot safely cross an asynchronous boundary when events can be generated on every cycle.

———————————

Chapter C3
Activity Monitor Unit

This chapter describes the *Activity Monitor Unit* (AMU).

It contains the following sections:

C3.1 About the AMU

The Cortex-A75 core includes activity monitoring. It has features in common with performance monitoring, but is intended for system management use whereas performance monitoring is aimed at user and debug applications.

The activity monitors provide useful information for system power management and persistent monitoring. The activity monitors are read-only in operation and their configuration is limited to the highest Exception level implemented.

The Cortex-A75 core implements five counters, 0-4, and activity monitoring is only implemented in AArch64.

C3.2 Accessing the activity monitors

The activity monitors can be accessed by:

- Core system registers.
- Memory-mapped access using the debug APB interface.

C3.2.1 Access enable bit

The access enable bit for traps on accesses to activity monitor registers is required at EL2 and EL3.

In the Cortex-A75 core, the CPUAMEN[4] bit in registers ACTLR_EL2 and ACTLR_EL3 controls the activity monitor registers enable.

Note

In the Cortex-A75 core, the CPUAMEN[4] bit is RES0 in ACTLR (S) and HACTLR. Activity monitors are not implemented in AArch32.

C3.2.2 System register access

The core implements activity monitoring in AArch64 and the activity monitors can be accessed using the `MRS` and `MSR` instructions.

C3.2.3 External memory-mapped access

Activity monitors can also be memory-mapped accessed from the APB debug interface.

In this case, the AMU registers just provide debug information and are read-only.

C3.3 AMU counters

The Cortex-A75 core implements five counters, 0-4. The activity monitor counters, CPUAMEVCNTR0-4, have the following characteristics:

- All events are counted in 64-bit wrapping counters that overflow when they wrap. There is no support for overflow status indication or interrupts.
- Counters monitoring cycle events do not increment when the core is in WFI or WFE state.
- Events 0, 1, and 2 are fixed and the CPUAMEVTYPER<n> evtCount bits are read-only.
- Events 3 and 4 are programmable at the highest Exception level implemented and the CPUAMEVTYPER<n> evtCount bits are read-write.

C3.4 AMU events

The following table describes the counters that are implemented in the Cortex-A75 core and the mapping to fixed and programmable events.

Table C3-1 Mapping of counters to fixed and programmable events

Activity monitor counter <n>	Event type	Event	Event number	Description
0	Fixed	Cycles at core frequency	`0x11`	Cycles count.
1	Fixed	Cycles at constant frequency	`0xEF`	This counter is used to replicate the generic system counter that is incremented on a constant basis, and not incremented depending on the PE frequency core.
2	Fixed	Instructions retired	`0x08`	Instruction architecturally executed. This counter increments for every instruction executed architecturally, including instructions that fail their condition code check.
3	Programmable	The events that can be selected for this counter are detailed in *Table C3-2 Programmable events* on page C3-569.		
4	Programmable	The events that can be selected for this counter are detailed in *Table C3-2 Programmable events* on page C3-569.		

The following table describes the possible events for counters 3 and 4.

Table C3-2 Programmable events

Event	Event number	Description
L1D_CACHE	`0x4`	L1 data cache access. This event counts read, write, and prefetch accesses to the L1 data cache. This includes non-cacheable speculative reads which do not have cacheability attributes yet. *Cache Maintenance Operation* (CMO) accesses are excluded.
L2D_CACHE	`0x16`	L2 data cache access. This event counts data reads, instruction reads, and prefetches that hit. Snoops are not counted.
L2D_CACHE_REFILL	`0x17`	L2 data cache refill. This event counts reads and prefetches that allocate a new linefill buffer entry. The reads and prefetches that fold into a prefetch are not counted.
BUS_ACCESS	`0x19`	Bus access. This event counts any move of data that is received from or sent to the SCU.

Table C3-2 Programmable events (continued)

Event	Event number	Description
STALL_FRONTEND	0x23	No operation issued because of the front end. The counter counts every cycle counted by the CPU_CYCLES event on which no operation was issued because there are no operations coming from the instruction side available to issue for this core. Flush windows are excluded.
STALL_BACKEND	0x24	No operation issued because of the back end. The counter counts every cycle counted by the CPU_CYCLES event on which no operation was issued because the rename stage cannot accept any instructions coming from the decoder stage. Flush windows are excluded.

Chapter C4
Embedded Trace Macrocell

This chapter describes the ETM for the Cortex-A75 core.

It contains the following sections:

C4.1 About the ETM

The ETM trace unit is a module that performs real-time instruction flow tracing based on the ETMv4 architecture. The ETM is a CoreSight component, and is an integral part of the Arm Real-time Debug solution, DS-5 Development Studio.

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

C4.2 ETM trace unit generation options and resources

The following table shows the trace generation options implemented in the Cortex-A75 ETM trace unit.

Table C4-1 ETM trace unit generation options implemented

Description	Configuration
Instruction address size in bytes	8
Data address size in bytes	0
Data value size in bytes	0
Virtual Machine ID size in bytes	4
Context ID size in bytes	4
Support for conditional instruction tracing	Not implemented
Support for tracing of data	Not implemented
Support for tracing of load and store instructions as P0 elements	Not implemented
Support for cycle counting in the instruction trace	Implemented
Support for branch broadcast tracing	Implemented
Number of events supported in the trace	4
Return stack support	Implemented
Tracing of SError exception support	Implemented
Instruction trace cycle counting minimum threshold	4
Size of Trace ID	7 bits
Synchronization period support	Read-write
Global timestamp size	64 bits
Number of cores available for tracing	1
ATB trigger support	Implemented
Low power behavior override	Implemented
Stall control support	Implemented
Support for overflow avoidance	Not implemented
Support for using CONTEXTIDR_EL2 in VMID comparator	Implemented

The following table shows the resources implemented in the Cortex-A75 ETM trace unit.

Table C4-2 ETM trace unit resources implemented

Description	Configuration
Number of resource selection pairs implemented	8
Number of external input selectors implemented	4
Number of external inputs implemented	32, 4 CTI + 28 PMU
Number of counters implemented	2
Reduced function counter implemented	Not implemented

Table C4-2 ETM trace unit resources implemented (continued)

Description	Configuration
Number of sequencer states implemented	4
Number of Virtual Machine ID comparators implemented	1
Number of Context ID comparators implemented	1
Number of address comparator pairs implemented	4
Number of single-shot comparator controls	1
Number of core comparator inputs implemented	0
Data address comparisons implemented	Not implemented
Number of data value comparators implemented	0

C4.3 ETM trace unit functional description

This section describes the functionality of the ETM trace unit.

The following figure shows the main functional blocks of the ETM trace unit.

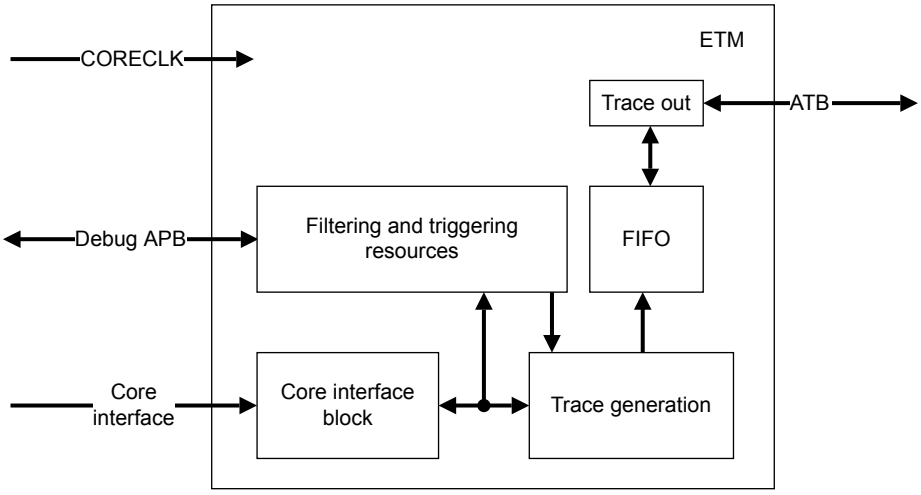

Figure C4-1 ETM functional blocks

Core interface
This block monitors the behavior of the core and generates P0 elements that are essentially executed branches and exceptions traced in program order.

Trace generation
The trace generation block generates various trace packets based on P0 elements.

Filtering and triggering resources
You can limit the amount of trace data generated by the ETM through the process of filtering.

For example, generating trace only in a certain address range. More complicated logic analyzer style filtering options are also available.

The ETM trace unit can also generate a trigger that is a signal to the trace capture device to stop capturing trace.

FIFO
The trace generated by the ETM trace unit is in a highly-compressed form.

The FIFO enables trace bursts to be flattened out. When the FIFO becomes full, the FIFO signals an overflow. The trace generation logic does not generate any new trace until the FIFO is emptied. This causes a gap in the trace when viewed in the debugger.

Trace out
Trace from FIFO is output on the AMBA ATB interface.

C4.4 Resetting the ETM

The reset for the ETM trace unit is the same as a Cold reset for the core.

The ETM trace unit is not reset when Warm reset is applied to the core so that tracing through Warm core reset is possible.

If the ETM trace unit is reset, tracing stops until the ETM trace unit is reprogrammed and re-enabled. However, if the core is reset using Warm reset, the last few instructions provided by the core before the reset might not be traced.

C4.5 Programming and reading ETM trace unit registers

You program and read the ETM trace unit registers using the Debug APB interface.

The core does not have to be in debug state when you program the ETM trace unit registers.

When you are programming the ETM trace unit registers, you must enable all the changes at the same time. Otherwise, if you program the counter, it might start to count based on incorrect events before the correct setup is in place for the trigger condition.

To disable the ETM trace unit, use the TRCPRGCTLR.EN bit.

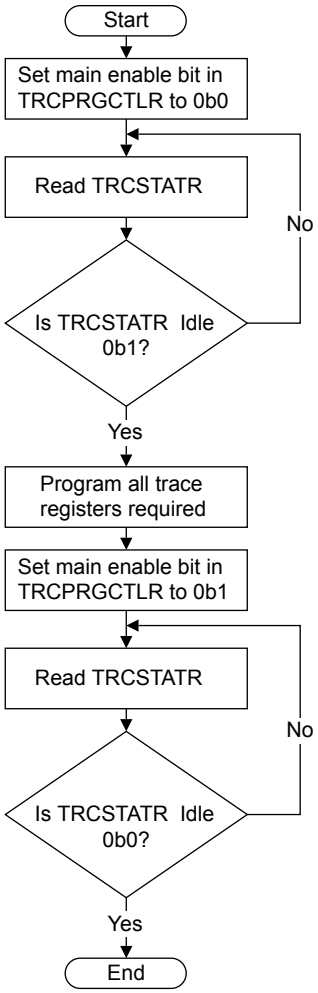

Figure C4-2 Programming ETM trace unit registers

C4.6 ETM trace unit register interfaces

The Cortex-A75 core supports only memory-mapped interface to trace registers.

See the *Arm® Embedded Trace Macrocell Architecture Specification ETMv4* for information on the behaviors on register accesses for different trace unit states and the different access mechanisms.

C4.7 Interaction with the PMU and Debug

This section describes the interaction with the PMU and the effect of debug double lock on trace register access.

Interaction with the PMU

The Cortex-A75 core includes a PMU that enables events, such as cache misses and instructions executed, to be counted over a period of time.

The PMU and ETM trace unit function together.

Use of PMU events by the ETM trace unit

The PMU architectural events described in *C2.3 PMU events* on page C2-554 are available to the ETM trace unit through the extended input facility.

See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile* for more information on PMU events.

The ETM trace unit uses four extended external input selectors to access the PMU events. Each selector can independently select one of the PMU events, that are then active for the cycles where the relevant events occur. These selected events can then be accessed by any of the event registers within the ETM trace unit. The PMU event table describes the PMU events.

Part D
Debug registers

Chapter D1
AArch32 debug registers

This chapter describes the debug registers in the AArch32 Execution state and shows examples of how to use them.

It contains the following sections:

D1.1 AArch32 debug register summary

The following table summarizes the 32-bit and 64-bit debug control registers that are accessible in the AArch32 Execution state from the internal CP14 interface. These registers are accessed by the MCR and MRC instructions in the order of CRn, op2, CRm, Op1 or MCRR and MRRC instructions in the order of CRm, Op1.

For those registers not described in this chapter, see the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

Table D1-1 AArch32 debug register summary

CRn	Op2	CRm	Op1	Name	Type	Reset	Description
c0	0	c0	0	DBGDIDR	RO	0x3518D000	*D1.4 DBGDIDR, Debug ID Register* on page D1-588
c0	0	c1	0	DBGDSCRint	RO	000X0000[f]	Debug Status and Control Register, Internal View
c0	0	c2	0	DBGDCCINT	RW	0x00000000	Debug Comms Channel Interrupt Enable Register
c0	0	c5	0	DBGDTRTXint	WO	-	Debug Data Transfer Register, Transmit, Internal View
c0	0	c5	0	DBGDTRRXint	RO	0x00000000	Debug Data Transfer Register, Receive, Internal View
c0	0	c6	0	DBGWFAR[g]	RW	-	Watchpoint Fault Address Register, RES0
c0	0	c7	0	DBGVCR	RW	0x00000000	Debug Vector Catch Register
c0	2	c0	0	DBGDTRRXext	RW	0x00000000	Debug Data Transfer Register, Receive, External View
c0	2	c2	0	DBGDSCRext	RW	000X0000[f]	Debug Status and Control Register, External View
c0	2	c3	0	DBGDTRTXext	RW	0x00000000	Debug Data Transfer Register, Transmit, External View
c0	2	c6	0	DBGOSECCR	RW	0x00000000	Debug OS Lock Exception Catch Control Register
c0	4	c0	0	DBGBVR0	RW	XXXXXXXX[h]	Debug Breakpoint Value Register 0
c0	4	c1	0	DBGBVR1	RW	XXXXXXXX[h]	Debug Breakpoint Value Register 1
c0	4	c2	0	DBGBVR2	RW	XXXXXXXX[h]	Debug Breakpoint Value Register 2
c0	4	c3	0	DBGBVR3	RW	XXXXXXXX[h]	Debug Breakpoint Value Register 3
c0	4	c4	0	DBGBVR4	RW	XXXXXXXX[h]	Debug Breakpoint Value Register 4
c0	4	c5	0	DBGBVR5	RW	XXXXXXXX[h]	Debug Breakpoint Value Register 5
c0	5	c0	0	DBGBCR0	RW	00XXXXXX[i]	Debug Breakpoint Control Register 0
c0	5	c1	0	DBGBCR1	RW	00XXXXXX[i]	Debug Breakpoint Control Register 1
c0	5	c2	0	DBGBCR2	RW	00XXXXXX[i]	Debug Breakpoint Control Register 2
c0	5	c3	0	DBGBCR3	RW	00XXXXXX[i]	Debug Breakpoint Control Register 3
c0	5	c4	0	DBGBCR4	RW	00XXXXXX[j]	Debug Breakpoint Control Register 4
c0	5	c5	0	DBGBCR5	RW	00XXXXXX[j]	Debug Breakpoint Control Register 5
c0	6	c0	0	DBGWVR0	RW	XXXXXXXX[h]	Debug Watchpoint Value Register 0
c0	6	c1	0	DBGWVR1	RW	XXXXXXXX[h]	Debug Watchpoint Value Register 1
c0	6	c2	0	DBGWVR2	RW	XXXXXXXX[h]	Debug Watchpoint Value Register 2
c0	6	c3	0	DBGWVR3	RW	XXXXXXXX[h]	Debug Watchpoint Value Register 3
c0	7	c0	0	DBGWCR0	RW	XXXXXXXX[k]	Watchpoint Control Register 0

Table D1-1 AArch32 debug register summary (continued)

CRn	Op2	CRm	Op1	Name	Type	Reset	Description
c0	7	c1	0	DBGWCR1	RW	xxxxxxxx[k]	Watchpoint Control Register 1
c0	7	c2	0	DBGWCR2	RW	xxxxxxxx[k]	Watchpoint Control Register 2
c0	7	c3	0	DBGWCR3	RW	xxxxxxxx[k]	Watchpoint Control Register 3
c1	0	c0	0	DBGDRAR[31:0]	RO	-	Debug ROM Address Register
-	-	c1	-	DBGDRAR[63:0]	RO	-	
c1	1	c4	0	DBGBXVR4	RW	xxxxxxxx[l]	Debug Breakpoint Extended Value Register 4
c1	1	c5	0	DBGBXVR5	RW	xxxxxxxx[l]	Debug Breakpoint Extended Value Register 5
c1	4	c0	0	DBGOSLAR	WO	-	Debug OS Lock Access Register
c1	4	c1	0	DBGOSLSR	RO	0x0000000A	Debug OS Lock Status Register
c1	4	c3	0	DBGOSDLR	RW	0x00000000	Debug OS Double Lock Register
c1	4	c4	0	DBGPRCR	RW	[m]	Debug Power/Reset Control Register
c2	2	c0	0	DBGDSAR[31:0]	RO	-	Debug Self Address Register RES0
-	0	c2	-	DBGDSAR[63:0][n]	RO	-	
c7	7	c0	0	DBGDEVID2	RO	0x00000000	Debug Device ID Register 2, RES0
c7	7	c1	0	DBGDEVID1	RO	0x00000000	*D1.3 DBGDEVID1, Debug Device ID Register 1* on page D1-587
c7	7	c2	0	DBGDEVID	RO	0x00110F10	*D1.2 DBGDEVID, Debug Device ID Register* on page D1-586
c7	6	c8	0	DBGCLAIMSET	RW	0x00110F00	Debug Claim Tag Set Register
c7	6	c9	0	DBGCLAIMCLR	RW	0x00000000	Debug Claim Tag Clear Register
c7	6	c14	0	DBGAUTHSTATUS	RO	0x000000AA	Debug Authentication Status Register

f The actual reset value is 32'b0000000000000x110000000000000000.
g Previously returned information about the address of the instruction that accessed a watchpoint address. This register is now deprecated and is RES0.
h The actual reset value is {30{1'bx}},2'b0
i The actual reset value is 32'b000000000x0x0x0xxx0000xxxxx00xx0.
j The actual reset value is 32'b00000000xxxx0x0xxx0000xxxxx00xx0.
k The actual reset value is 32'b000xxxxx000x0x0xxxxxxxxxxxxxxxx0.
l The actual reset value is 32'hxxxxxxxx.
m The actual reset value is 31'b0000000000000000000000000000000, EDPRCR.COREPURQ.
n Previously defined the offset from the base address defined in DBGDRAR of the physical base address of the debug registers for the core. This register is now deprecated and RES0.

D1.2 DBGDEVID, Debug Device ID Register

The DBGDEVID specifies the version of the Debug architecture implemented and some features of the debug implementation.

Bit field descriptions

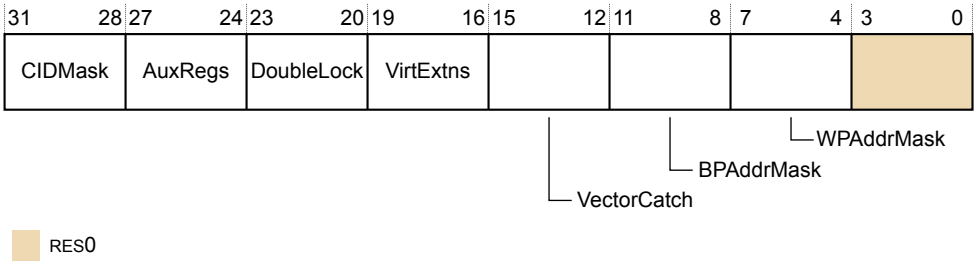

Figure D1-1 DBGDEVID bit assignments

CIDMask, [31:28]
Specifies the level of support for the Context ID matching breakpoint masking capability. This value is:

`0x0` Context ID masking is not implemented.

AuxRegs, [27:24]
Specifies support for the Debug External Auxiliary Control Register. This value is:

`0x0` None supported.

DoubleLock, [23:20]
Specifies support for the Debug OS Double Lock Register. This value is:

`0x1` The core supports Debug OS Double Lock Register.

VirtExtns, [19:16]
Specifies whether EL2 is implemented. This value is:

`0x1` The core implements EL2.

VectorCatch, [15:12]
Defines the form of the vector catch event implemented. This value is:

`0x0` The core implements address matching form of vector catch.

BPAddrMask, [11:8]
Indicates the level of support for the *Immediate Virtual Address* (IVA) matching breakpoint masking capability. This value is:

`0xF` Breakpoint address masking not implemented. DBGBCRn[28:24] are RES0.

WPAddrMask, [7:4]
Indicates the level of support for the DVA matching watchpoint masking capability. This value is:

`0x1` Watchpoint address mask implemented.

[3:0]
Reserved, RES0.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

D1.3 DBGDEVID1, Debug Device ID Register 1

The DBGDEVID1 adds to the information given by the DBGDIDR by describing other features of the debug implementation.

Bit field descriptions

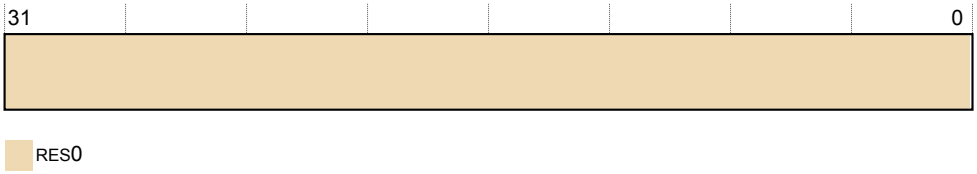

Figure D1-2 DBGDEVID1 bit assignments

RES0, [31:0]

RES0 Reserved.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

D1.4 DBGDIDR, Debug ID Register

The DBGDIDR specifies the version of the Debug architecture that is implemented and some features of the debug implementation.

Bit field descriptions

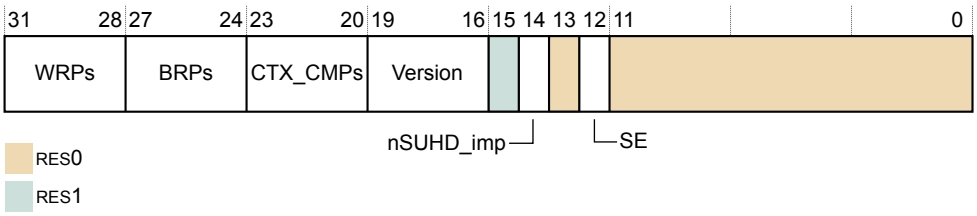

Figure D1-3 DBGDIDR bit assignments

WRPs, [31:28]
The number of *Watchpoint Register Pairs* (WRPs) implemented. The number of implemented WRPs is one more than the value of this field. The value is:

`0x3` The core implements 4 WRPs.

This field has the same value as ID_AA64DFR0_EL1.WRPs.

BRPs, [27:24]
The number of *Breakpoint Register Pairs* (BRPs) implemented. The number of implemented BRPs is one more than the value of this field. The value is:

`0x5` The core implements 6 BRPs.

This field has the same value as ID_AA64DFR0_EL1.BRPs.

CTX_CMPs, [23:20]
The number of BRPs that can be used for Context matching. This is one more than the value of this field. The value is:

`0x1` The core implements two Context matching breakpoints, breakpoints 4 and 5.

This field has the same value as ID_AA64DFR0_EL1.CTX_CMPs.

Version, [19:16]
The Debug architecture version.

`0x8` The core implements the Armv8.2 Debug architecture.

RES1, [15]

RES1 Reserved.

nSUHD_imp, [14]
Secure User Halting Debug not implemented bit. The value is:

`1` The core does not implement Secure User Halting Debug.

RES0, [13]

RES0 Reserved.

SE, [12]
EL3 implemented. The value is:

`1` The core implements EL3.

RES0, [11:0]

RES0 Reserved.

Chapter D2
AArch64 debug registers

This chapter describes the debug registers in the AArch64 Execution state and shows examples of how to use them.

It contains the following sections:

D2.1 AArch64 debug register summary

This section summarizes the debug control registers that are accessible in the AArch64 Execution state.

These registers, listed in the following table, are accessed by the `MRS` and `MSR` instructions in the order of Op0, CRn, Op1, CRm, Op2.

See *D3.1 Memory-mapped debug register summary* on page D3-602 for a complete list of registers accessible from the external debug interface. The 64-bit registers cover two addresses on the external memory interface. For those registers not described in this chapter, see the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

Table D2-1 AArch64 debug register summary

Name	Type	Reset	Width	Description
OSDTRRX_EL1	RW	`0x00000000`	32	Debug Data Transfer Register, Receive, External View
DBGBVR0_EL1	RW	-	64	Debug Breakpoint Value Register 0
DBGBCR0_EL1	RW	UNK	32	*D2.2 DBGBCRn_EL1, Debug Breakpoint Control Registers, EL1* on page D2-594
DBGWVR0_EL1	RW	-	64	Debug Watchpoint Value Register 0
DBGWCR0_EL1	RW	UNK	32	*D2.4 DBGWCRn_EL1, Debug Watchpoint Control Registers, EL1* on page D2-598
DBGBVR1_EL1	RW	-	64	Debug Breakpoint Value Register 1
DBGBCR1_EL1	RW	UNK	32	*D2.2 DBGBCRn_EL1, Debug Breakpoint Control Registers, EL1* on page D2-594
DBGWVR1_EL1	RW	-	64	Debug Watchpoint Value Register 1
DBGWCR1_EL1	RW	UNK	32	*D2.4 DBGWCRn_EL1, Debug Watchpoint Control Registers, EL1* on page D2-598
MDCCINT_EL1	RW	`0x00000000`	32	Monitor Debug Comms Channel Interrupt Enable Register
MDSCR_EL1	RW	-	32	Monitor Debug System Register
DBGBVR2_EL1	RW	-	64	Debug Breakpoint Value Register 2
DBGBCR2_EL1	RW	UNK	32	*D2.2 DBGBCRn_EL1, Debug Breakpoint Control Registers, EL1* on page D2-594
DBGWVR2_EL1	RW	-	64	Debug Watchpoint Value Register 2
DBGWCR2_EL1	RW	UNK	32	*D2.4 DBGWCRn_EL1, Debug Watchpoint Control Registers, EL1* on page D2-598
OSDTRTX_EL1	RW	-	32	Debug Data Transfer Register, Transmit, External View
DBGBVR3_EL1	RW	-	64	Debug Breakpoint Value Register 3
DBGBCR3_EL1	RW	UNK	32	*D2.2 DBGBCRn_EL1, Debug Breakpoint Control Registers, EL1* on page D2-594
DBGWVR3_EL1	RW	-	64	Debug Watchpoint Value Register 3
DBGWCR3_EL1	RW	UNK	32	*D2.4 DBGWCRn_EL1, Debug Watchpoint Control Registers, EL1* on page D2-598
DBGBVR4_EL1	RW	-	64	Debug Breakpoint Value Register 4

Table D2-1 AArch64 debug register summary (continued)

Name	Type	Reset	Width	Description
DBGBCR4_EL1	RW	UNK	32	*D2.2 DBGBCRn_EL1, Debug Breakpoint Control Registers, EL1* on page D2-594
DBGBVR5_EL1	RW	-	64	Debug Breakpoint Value Register 5
DBGBCR5_EL1	RW	UNK	32	*D2.2 DBGBCRn_EL1, Debug Breakpoint Control Registers, EL1* on page D2-594
OSECCR_EL1	RW	0x00000000	32	Debug OS Lock Exception Catch Register
MDCCSR_EL0	RO	0x00000000	32	Monitor Debug Comms Channel Status Register
DBGDTR_EL0	RW	0x00000000	64	Debug Data Transfer Register, half-duplex
DBGDTRTX_EL0	WO	0x00000000	32	Debug Data Transfer Register, Transmit, Internal View
DBGDTRRX_EL0	RO	0x00000000	32	Debug Data Transfer Register, Receive, Internal View
DBGVCR32_EL2	RW	-	32	Debug Vector Catch Register
MDRAR_EL1	RO	-	64	Debug ROM Address Register. This register is reserved, RES0
OSLAR_EL1	WO	-	32	Debug OS Lock Access Register
OSLSR_EL1	RO	0x0000000A	32	Debug OS Lock Status Register
OSDLR_EL1	RW	0x00000000	32	Debug OS Double Lock Register
DBGPRCR_EL1	RW	-	32	Debug Power/Reset Control Register
DBGCLAIMSET_EL1	RW	0x000000FF	32	Debug Claim Tag Set Register
DBGCLAIMCLR_EL1	RW	0x00000000	32	Debug Claim Tag Clear Register
DBGAUTHSTATUS_EL1	RO	0x000000AA	32	Debug Authentication Status Register

D2.2 DBGBCRn_EL1, Debug Breakpoint Control Registers, EL1

The DBGBCR*n*_EL1 holds control information for a breakpoint. Each DBGBVR_EL1 is associated with a DBGBCR_EL1 to form a *Breakpoint Register Pair* (BRP). DBGBVR*n*_EL1 is associated with DBGBCR*n*_EL1 to form BRP*n*. The range of *n* for DBGBCR*n*_EL1 is 0 to 5.

Bit field descriptions

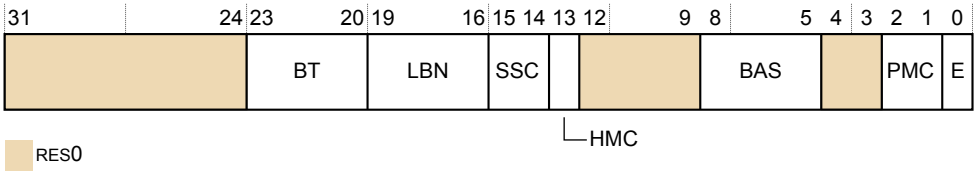

Figure D2-1 DBGBCR*n*_EL1

RES0, [31:24]

RES0 Reserved.

BT, [23:20]

Breakpoint Type. This field controls the behavior of Breakpoint debug event generation. This includes the meaning of the value held in the associated DBGBVR*n*_EL1, indicating whether it is an instruction address match or mismatch, or a Context match. It also controls whether the breakpoint is linked to another breakpoint. The possible values are:

`0b0000` Unlinked instruction address match.

`0b0001` Linked instruction address match.

`0b0010` Unlinked Context ID match.

`0b0011` Linked Context ID match.

`0b0100` Unlinked instruction address mismatch.

`0b0101` Linked instruction address mismatch.

`0b0110` Unlinked CONTEXTIDR_EL1 match.

`0b0111` Linked CONTEXTIDR_EL1 match.

`0b1000` Unlinked VMID match.

`0b1001` Linked VMID match.

`0b1010` Unlinked VMID + Conext ID match.

`0b1011` Linked VMID + Context ID match.

`0b1100` Unlinked CONTEXTIDR_EL2 match.

`0b1101` Linked CONTEXTIDR_EL2 match.

`0b1110` Unlinked Full Context ID match.

`0b1111` Linked Full Context ID match.

The field break down is:

- BT[3:1]: Base type. If the breakpoint is not context-aware, these bits are RES0. Otherwise, the possible values are:

  `0b000` Match address. DBGBVR*n*_EL1 is the address of an instruction.

  `0b001` Match context ID. DBGBVR*n*_EL1[31:0] is a context ID.

`0b010` Address mismatch. Mismatch address. Behaves as type 0b000 if either:
— In an AArch64 translation regime.
— Halting debug-mode is enabled and halting is allowed.

Otherwise, DBGBVR*n*_EL1 is the address of an instruction to be stepped.

`0b011` Match CONTEXTIDR_EL1. DBGBVR*n*_EL1[31:0] is a context ID.

`0b100` Match VMID. DBGBVR*n*_EL1[47:32] is a VMID.

`0b101` Match VMID and CONTEXTIDR_EL1. DBGBVR*n*_EL1[31:0] is a context ID, and DBGBVR*n*_EL1[47:32] is a VMID.

`0b110` Match CONTEXTIDR_EL2. DBGBVR*n*_EL1[63:32] is a context ID.

`0b111` Match CONTEXTIDR_EL1 and CONTEXTIDR_EL2. DBGBVR*n*_EL1[31:0] and DBGBVR*n*_EL1[63:32] are Context IDs.

- BT[0]: Enable linking.

LBN, [19:16]
Linked breakpoint number. For Linked address matching breakpoints, this specifies the index of the Context-matching breakpoint linked to.

SSC, [15:14]
Security State Control. Determines the security states that a breakpoint debug event for breakpoint *n* is generated.

This field must be interpreted with the *Higher Mode Control* (HMC), and *Privileged Mode Control* (PMC), fields to determine the mode and security states that can be tested.

See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile* for possible values of the HMC and PMC fields.

HMC, [13]
Hyp Mode Control bit. Determines the debug perspective for deciding when a breakpoint debug event for breakpoint *n* is generated.

This bit must be interpreted with the SSC and PMC fields to determine the mode and security states that can be tested.

See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile* for possible values of the SSC and PMC fields.

RES0, [12:9]

RES0 Reserved.

BAS, [8:5]
Byte Address Select. Defines which half-words a regular breakpoint matches, regardless of the instruction set and execution state. A debugger must program this field as follows:

`0x3` Match the T32 instruction at DBGBVR*n*_EL1.

`0xC` Match the T32 instruction at DBGBVR*n*+2_EL1.

`0xF` Match the A64 or A32 instruction at DBGBVR*n*_EL1, or context match.

All other values are reserved.

The Armv8-A architecture does not support direct execution of Java bytecodes. BAS[3] and BAS[1] ignore writes and on reads return the values of BAS[2] and BAS[0] respectively.

See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile* for more information on how the BAS field is interpreted by hardware.

RES0, [4:3]

RES0 Reserved.

PMC, [2:1]

Privileged Mode Control. Determines the exception level or levels that a breakpoint debug event for breakpoint *n* is generated.

This field must be interpreted with the SSC and HMC fields to determine the mode and security states that can be tested.

See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile* for possible values of the SSC and HMC fields.

Bits[2:1] have no effect for accesses made in Hyp mode.

E, [0]

Enable breakpoint. This bit enables the BRP:

`0` BRP disabled.

`1` BRP enabled.

A BRP never generates a breakpoint debug event when it is disabled.

The value of DBGBCR*n*_EL1.E is UNKNOWN on reset. A debugger must ensure that DBGBCR*n*_EL1.E has a defined value before it enables debug.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

D2.3 DBGCLAIMSET_EL1, Debug Claim Tag Set Register, EL1

The DBGCLAIMSET_EL1 is used by software to set CLAIM bits to 1.

Bit field descriptions

The following figure shows the DBGCLAIMSET_EL1 bit assignments.

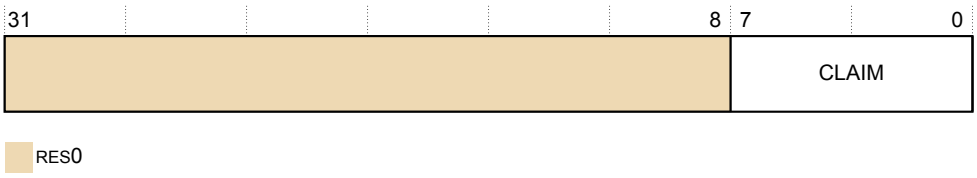

Figure D2-2 DBGCLAIMSET_EL1 bit assignments

RES0, [31:8]

RES0 Reserved.

CLAIM, [7:0]

Claim set bits.

Writing a 1 to one of these bits sets the corresponding CLAIM bit to 1. This is an indirect write to the CLAIM bits.

A single write operation can set multiple bits to 1. Writing 0 to one of these bits has no effect.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

D2.4 DBGWCRn_EL1, Debug Watchpoint Control Registers, EL1

The DBGWCR*n*_EL1 holds control information for a watchpoint. Each DBGWCR_EL1 is associated with a DBGWVR_EL1 to form a *Watchpoint Register Pair* (WRP). DBGWCR*n*_EL1 is associated with DBGWVR*n*_EL1 to form WRP*n*. The range of *n* for DBGBCR*n*_EL1 is 0 to 3.

Bit field descriptions

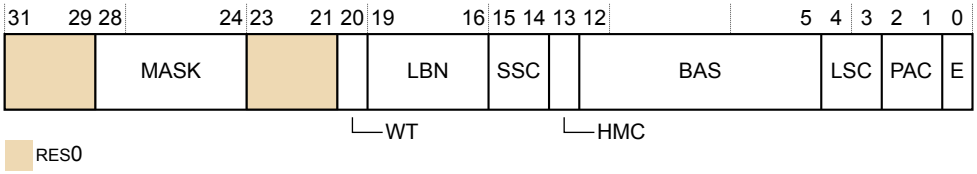

Figure D2-3 DBGWCR_EL1

RES0, [31:29]

RES0 Reserved.

MASK, [28:24]

Address mask. Only objects up to 2GB can be watched using a single mask.

`0b0000` No mask.

`0b0001` Reserved.

`0b0010` Reserved.

Other values mask the corresponding number of address bits, from `0b00011` masking 3 address bits (`0x00000007` mask for address) to `0b11111` masking 31 address bits (`0x7FFFFFFF` mask for address).

RES0, [23:21]

RES0 Reserved.

WT, [20]

Watchpoint type. Possible values are:

`0` Unlinked data address match.

`1` Linked data address match.

On Cold reset, the field reset value is architecturally UNKNOWN.

LBN, [19:16]

Linked breakpoint number. For Linked data address watchpoints, this specifies the index of the Context-matching breakpoint linked to.

On Cold reset, the field reset value is architecturally UNKNOWN.

SSC, [15:14]

Security state control. Determines the Security states under which a watchpoint debug event for watchpoint n is generated. This field must be interpreted along with the HMC and PAC fields.

On Cold reset, the field reset value is architecturally UNKNOWN.

HMC, [13]

Higher mode control. Determines the debug perspective for deciding when a watchpoint debug event for watchpoint n is generated. This field must be interpreted along with the SSC and PAC fields.

On Cold reset, the field reset value is architecturally UNKNOWN.

BAS, [12:5]

Byte address select. Each bit of this field selects whether a byte from within the word or double-word addressed by DBGWVR*n*_EL1 is being watched. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile* for more information.

LSC, [4:3]

Load/store access control. This field enables watchpoint matching on the type of access being made. The possible values are:

`0b01` Match instructions that load from a watchpoint address.

`0b10` Match instructions that store to a watchpoint address.

`0b11` Match instructions that load from or store to a watchpoint address.

All other values are reserved, but must behave as if the watchpoint is disabled. Software must not rely on this property as the behavior of reserved values might change in a future revision of the architecture.

Ignored if E is 0.

On Cold reset, the field reset value is architecturally UNKNOWN.

PAC, [2:1]

Privilege of access control. Determines the exception level or levels at which a watchpoint debug event for watchpoint n is generated. This field must be interpreted along with the SSC and HMCfields.

On Cold reset, the field reset value is architecturally UNKNOWN.

E, [0]

Enable watchpoint n. Possible values are:

`0` Watchpoint disabled.

`1` Watchpoint enabled.

On Cold reset, the field reset value is architecturally UNKNOWN.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

Chapter D3
Memory-mapped debug registers

This chapter describes the memory-mapped debug registers and shows examples of how to use them.

It contains the following sections:

D3.1 Memory-mapped debug register summary

The following table shows the offset address for the registers that are accessible from the external debug interface.

For those registers not described in this chapter, see the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

Table D3-1 Memory-mapped debug register summary

Offset	Name	Type	Width	Description
0x000-0x01C	-	-	-	Reserved
0x020	EDESR	RW	32	External Debug Event Status Register
0x024	EDECR	RW	32	External Debug Execution Control Register
0x028-0x02C	-	-	-	Reserved
0x030	EDWAR[31:0]	RO	64	External Debug Watchpoint Address Register
0x034	EDWAR[63:32]			
0x038-0x07C	-	-	-	Reserved
0x080	DBGDTRRX_EL0	RW	32	Debug Data Transfer Register, Receive
0x084	EDITR	WO	32	External Debug Instruction Transfer Register
0x088	EDSCR	RW	32	External Debug Status and Control Register
0x08C	DBGDTRTX_EL0	WO	32	Debug Data Transfer Register, Transmit
0x090	EDRCR	WO	32	*D3.14 EDRCR, External Debug Reserve Control Register* on page D3-618
0x094	EDACR	RW	32	Reserved
0x098	EDECCR	RW	32	External Debug Exception Catch Control Register
0x09C	-	-	-	Reserved
0x0A0	-	-	-	Reserved
0x0A4	-	-	-	Reserved
0x0A8	-	-	-	Reserved
0x0AC	-	-	-	Reserved
0x0B0-0x2FC	-	-	-	Reserved
0x300	OSLAR_EL1	WO	32	OS Lock Access Register
0x304-0x30C	-	-	-	Reserved
0x310	EDPRCR	RW	32	External Debug Power/Reset Control Register
0x314	EDPRSR	RO	32	External Debug Processor Status Register
0x318-0x3FC	-	-	-	Reserved
0x400	DBGBVR0_EL1[31:0]	RW	64	Debug Breakpoint Value Register 0
0x404	DBGBVR0_EL1[63:32]			
0x408	DBGBCR0_EL1	RW	32	*D2.2 DBGBCRn_EL1, Debug Breakpoint Control Registers, EL1* on page D2-594

Table D3-1 Memory-mapped debug register summary (continued)

Offset	Name	Type	Width	Description
0x40C	-	-	-	Reserved
0x410	DBGBVR1_EL1[31:0]	RW	64	Debug Breakpoint Value Register 1
0x414	DBGBVR1_EL1[63:32]			
0x418	DBGBCR1_EL1	RW	32	*D2.2 DBGBCRn_EL1, Debug Breakpoint Control Registers, EL1* on page D2-594
0x41C	-	-	-	Reserved
0x420	DBGBVR2_EL1[31:0]	RW	64	Debug Breakpoint Value Register 2
0x424	DBGBVR2_EL1[63:32]			
0x428	DBGBCR2_EL1	RW	32	*D2.2 DBGBCRn_EL1, Debug Breakpoint Control Registers, EL1* on page D2-594
0x42C	-	-	-	Reserved
0x430	DBGBVR3_EL1[31:0]	RW	64	Debug Breakpoint Value Register 3
0x434	DBGBVR3_EL1[63:32]			
0x438	DBGBCR3_EL1	RW	32	*D2.2 DBGBCRn_EL1, Debug Breakpoint Control Registers, EL1* on page D2-594
0x43C	-	-	-	Reserved
0x440	DBGBVR4_EL1[31:0]	RW	64	Debug Breakpoint Value Register 4
0x444	DBGBVR4_EL1[63:32]			
0x448	DBGBCR4_EL1	RW	32	*D2.2 DBGBCRn_EL1, Debug Breakpoint Control Registers, EL1* on page D2-594
0x44C	-	-	-	Reserved
0x450	DBGBVR5_EL1[31:0]	RW	64	Debug Breakpoint Value Register 5
0x454	DBGBVR5_EL1[63:32]			
0x458	DBGBCR5_EL1	RW	32	*D2.2 DBGBCRn_EL1, Debug Breakpoint Control Registers, EL1* on page D2-594
0x45C-0x7FC	-	-	-	Reserved
0x800	DBGWVR0_EL1[31:0]	RW	64	Debug Watchpoint Value Register 0
0x804	DBGWVR0_EL1[63:32]			
0x808	DBGWCR0_EL1	RW	32	*D2.4 DBGWCRn_EL1, Debug Watchpoint Control Registers, EL1* on page D2-598
0x80C	-	-	-	Reserved
0x810	DBGWVR1_EL1[31:0]	RW	64	Debug Watchpoint Value Register 1
0x814	DBGWVR1_EL1[63:32]			
0x818	DBGWCR1_EL1	RW	32	*D2.4 DBGWCRn_EL1, Debug Watchpoint Control Registers, EL1* on page D2-598
0x81C	-	-	-	Reserved

Table D3-1 Memory-mapped debug register summary (continued)

Offset	Name	Type	Width	Description
0x820	DBGWVR2_EL1[31:0]	RW	64	Debug Watchpoint Value Register 2
0x824	DBGWVR2_EL1[63:32]			
0x828	DBGWCR2_EL1	RW	32	*D2.4 DBGWCRn_EL1, Debug Watchpoint Control Registers, EL1* on page D2-598
0x82C	-	-	-	Reserved
0x830	DBGWVR3_EL1[31:0]	RW	64	Debug Watchpoint Value Register 0,
0x834	DBGWVR3_EL1[63:32]			
0x838	DBGWCR3_EL1	RW	32	*D2.4 DBGWCRn_EL1, Debug Watchpoint Control Registers, EL1* on page D2-598
0x83C-0xCFC	-	-	-	Reserved
0xD00	MIDR	RO	32	*B2.83 MIDR_EL1, Main ID Register, EL1* on page B2-408
0xD04-0xD1C	-	-	-	Reserved
0xD20	EDPFR[31:0]	RO	64	*B2.61 ID_AA64PFR0_EL1, AArch64 Processor Feature Register 0, EL1* on page B2-371
0xD24	EDPFR[63:32]			
0xD28	EDDFR[31:0]	RO	64	*B2.61 ID_AA64PFR0_EL1, AArch64 Processor Feature Register 0, EL1* on page B2-371
0xD2C	EDDFR[63:32]			
0xD60-0xEFC	-	-	-	Reserved
0xF00	-	-	-	Reserved
0xF04-0xF9C	-	-	-	Reserved
0xFA0	DBGCLAIMSET_EL1	RW	32	*D2.3 DBGCLAIMSET_EL1, Debug Claim Tag Set Register, EL1* on page D2-597
0xFA4	DBGCLAIMCLR_EL1	RW	32	Debug Claim Tag Clear Register
0xFA8	EDDEVAFF0	RO	32	*B1.75 MPIDR, Multiprocessor Affinity Register* on page B1-243
0xFAC	EDDEVAFF1	RO	32	External Debug Device Affinity Register 1
0xFB0	-	-	-	Reserved
0xFB4	-	-	-	Reserved
0xFB8	DBGAUTHSTATUS_EL1	RO	32	Debug Authentication Status Register
0xFBC	EDDEVARCH	RO	32	External Debug Device Architecture Register
0xFC0	EDDEVID2	RO	32	External Debug Device ID Register 2, RES0
0xFC4	EDDEVID1	RO	32	*D3.7 EDDEVID1, External Debug Device ID Register 1* on page D3-611
0xFC8	EDDEVID	RO	32	*D3.6 EDDEVID, External Debug Device ID Register 0* on page D3-610
0xFCC	EDDEVTYPE	RO	32	External Debug Device Type Register
0xFD0	EDPIDR4	RO	32	*D3.12 EDPIDR4, External Debug Peripheral Identification Register 4* on page D3-616
0xFD4-0xFDC	EDPIDR5-7	RO	32	*D3.13 EDPIDRn, External Debug Peripheral Identification Registers 5-7* on page D3-617

Table D3-1 Memory-mapped debug register summary (continued)

Offset	Name	Type	Width	Description
0xFE0	EDPIDR0	RO	32	*D3.8 EDPIDR0, External Debug Peripheral Identification Register 0* on page D3-612
0xFE4	EDPIDR1	RO	32	*D3.9 EDPIDR1, External Debug Peripheral Identification Register 1* on page D3-613
0xFE8	EDPIDR2	RO	32	*D3.10 EDPIDR2, External Debug Peripheral Identification Register 2* on page D3-614
0xFEC	EDPIDR3	RO	32	*D3.11 EDPIDR3, External Debug Peripheral Identification Register 3* on page D3-615
0xFF0	EDCIDR0	RO	32	*D3.2 EDCIDR0, External Debug Component Identification Register 0* on page D3-606
0xFF4	EDCIDR1	RO	32	*D3.3 EDCIDR1, External Debug Component Identification Register 1* on page D3-607
0xFF8	EDCIDR2	RO	32	*D3.4 EDCIDR2, External Debug Component Identification Register 2* on page D3-608
0xFFC	EDCIDR3	RO	32	*D3.5 EDCIDR3, External Debug Component Identification Register 3* on page D3-609

D3.2 EDCIDR0, External Debug Component Identification Register 0

The EDCIDR0 provides information to identify an external debug component.

Bit field descriptions

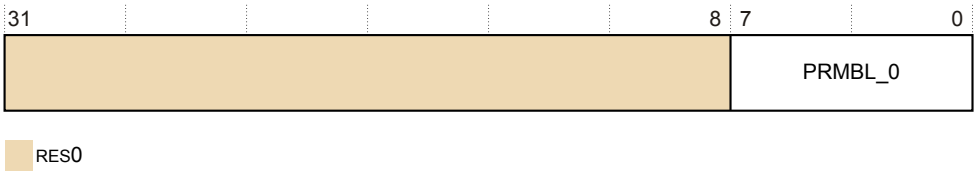

Figure D3-1 EDCIDR0 bit assignments

RES0, [31:8]

RES0 Reserved.

PRMBL_0, [7:0]

`0x0D` Preamble byte 0.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The EDCIDR0 can be accessed through the external debug interface, offset `0xFF0`.

D3.3 EDCIDR1, External Debug Component Identification Register 1

The EDCIDR1 provides information to identify an external debug component.

Bit field descriptions

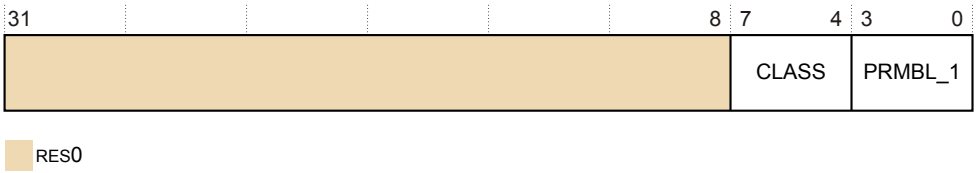

Figure D3-2 EDCIDR1 bit assignments

RES0, [31:8]

RES0 Reserved.

CLASS, [7:4]

`0x9` Debug component.

PRMBL_1, [3:0]

`0x0` Preamble.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The EDCIDR1 can be accessed through the external debug interface, offset `0xFF4`.

D3.4 EDCIDR2, External Debug Component Identification Register 2

The EDCIDR2 provides information to identify an external debug component.

Bit field descriptions

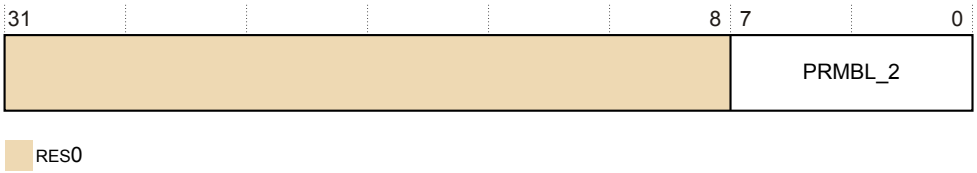

Figure D3-3 EDCIDR2 bit assignments

RES0, [31:8]

RES0 Reserved.

PRMBL_2, [7:0]

`0x05` Preamble byte 2.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The EDCIDR2 can be accessed through the external debug interface, offset `0xFF8`.

D3.5 EDCIDR3, External Debug Component Identification Register 3

The EDCIDR3 provides information to identify an external debug component.

Bit field descriptions

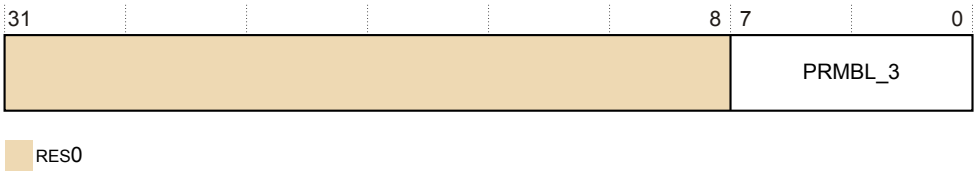

Figure D3-4 EDCIDR3 bit assignments

RES0, [31:8]

RES0 Reserved.

PRMBL_3, [7:0]

`0xB1` Preamble byte 3.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The EDCIDR3 can be accessed through the external debug interface, offset `0xFFC`.

D3.6 EDDEVID, External Debug Device ID Register 0

The EDDEVID provides extra information for external debuggers about features of the debug implementation.

Bit field descriptions

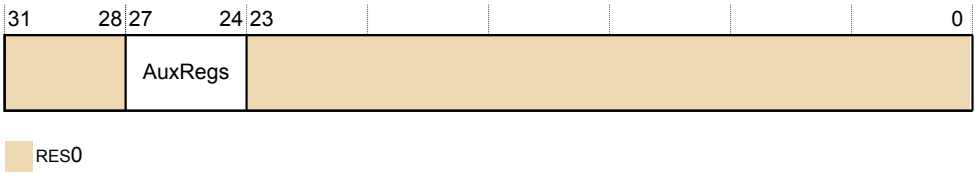

Figure D3-5 EDDEVID bit assignments

RES0, [31:28]

RES0 Reserved.

AuxRegs, [27:24]

Indicates support for Auxiliary registers:

`0x0` None supported.

RES0, [23:0]

RES0 Reserved.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The EDDEVID can be accessed through the external debug interface, offset `0xFC8`.

D3.7 EDDEVID1, External Debug Device ID Register 1

The EDDEVID1 provides extra information for external debuggers about features of the debug implementation.

Bit field descriptions

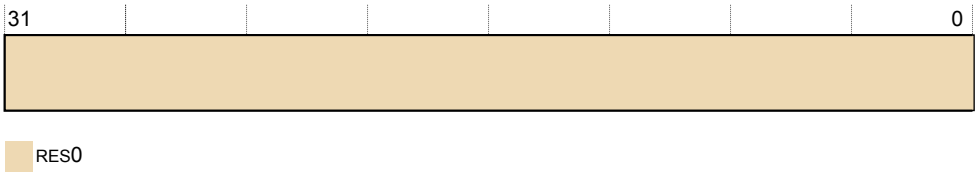

Figure D3-6 EDDEVID1 bit assignments

RES0, [31:0]

RES0 Reserved.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The EDDEVID1 can be accessed through the external debug interface, offset `0xFC4`.

D3.8 EDPIDR0, External Debug Peripheral Identification Register 0

The EDPIDR0 provides information to identify an external debug component.

Bit field descriptions

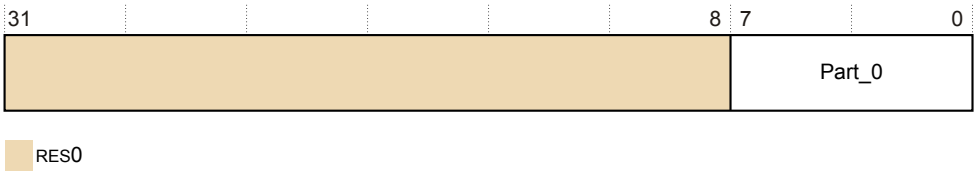

Figure D3-7 EDPIDR0 bit assignments

RES0, [31:8]

RES0 Reserved.

Part_0, [7:0]

`0x0A` Least significant byte of the debug part number.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The EDPIDR0 can be accessed through the external debug interface, offset `0xFE0`.

D3.9 EDPIDR1, External Debug Peripheral Identification Register 1

The EDPIDR1 provides information to identify an external debug component.

Bit field descriptions

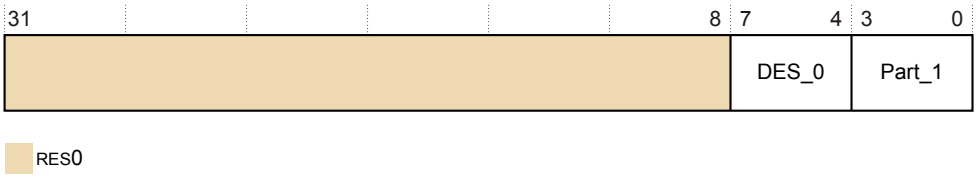

Figure D3-8 EDPIDR1 bit assignments

RES0, [31:8]

RES0 Reserved.

DES_0, [7:4]

`0xB` Arm Limited. This is the least significant nibble of JEP106 ID code.

Part_1, [3:0]

`0xD` Most significant nibble of the debug part number.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The EDPIDR1 can be accessed through the external debug interface, offset `0xFE4`.

D3.10 EDPIDR2, External Debug Peripheral Identification Register 2

The EDPIDR2 provides information to identify an external debug component.

Bit field descriptions

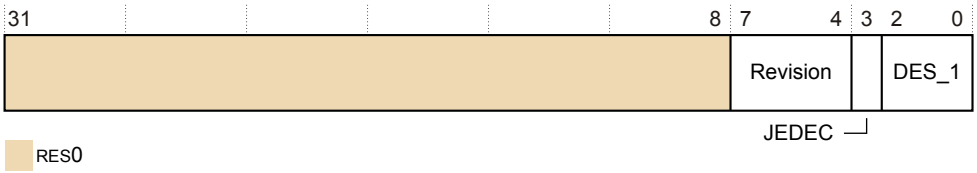

Figure D3-9 EDPIDR2 bit assignments

RES0, [31:8]

RES0 Reserved.

Revision, [7:4]

1 r2p1.

JEDEC, [3]

`0b1` RAO. Indicates a JEP106 identity code is used.

DES_1, [2:0]

`0b011` Arm Limited. This is the most significant nibble of JEP106 ID code.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The EDPIDR2 can be accessed through the external debug interface, offset `0xFE8`.

D3.11 EDPIDR3, External Debug Peripheral Identification Register 3

The EDPIDR3 provides information to identify an external debug component.

Bit field descriptions

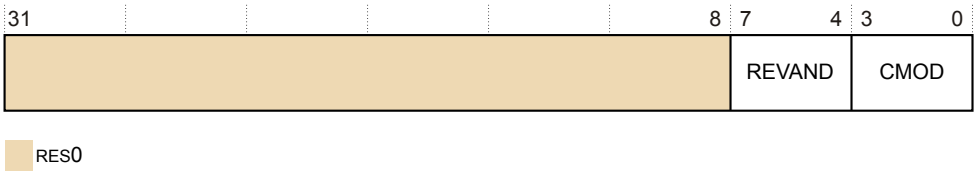

Figure D3-10 EDPIDR3 bit assignments

RES0, [31:8]

RES0 Reserved.

REVAND, [7:4]

`0x0` Part minor revision.

CMOD, [3:0]

`0x0` Customer modified.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The EDPIDR3 can be accessed through the external debug interface, offset `0xFEC`.

D3.12 EDPIDR4, External Debug Peripheral Identification Register 4

The EDPIDR4 provides information to identify an external debug component.

Bit field descriptions

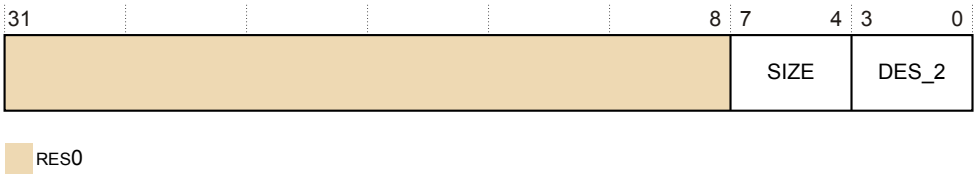

Figure D3-11 EDPIDR4 bit assignments

RES0, [31:8]

RES0 Reserved.

SIZE, [7:4]

`0x0` Size of the component. Log_2 the number of 4KB pages from the start of the component to the end of the component ID registers.

DES_2, [3:0]

`0x4` Arm Limited This is the least significant nibble JEP106 continuation code.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The EDPIDR4 can be accessed through the external debug interface, offset `0xFD0`.

D3.13 EDPIDRn, External Debug Peripheral Identification Registers 5-7

No information is held in the Peripheral ID5, Peripheral ID6, and Peripheral ID7 Registers.

They are reserved for future use and are RES0.

D3.14 EDRCR, External Debug Reserve Control Register

The EDRCR is part of the Debug registers functional group. This register is used to allow imprecise entry to Debug state and clear sticky bits in EDSCR.

Bit field descriptions

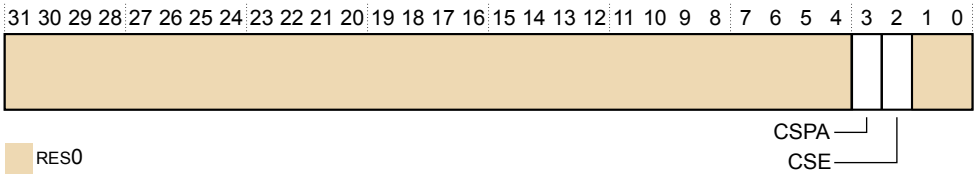

Figure D3-12 EDRCR bit assignments

RES0, [31:4]

RES0 Reserved.

CSPA, [3]

Clear Sticky Pipeline Advance. This bit is used to clear the EDSCR.PipeAdv bit to 0. The actions on writing to this bit are:

`0` No action.

`1` Clear the EDSCR.PipeAdv bit to 0.

CSE, [2]

Clear Sticky Error. Used to clear the EDSCR cumulative error bits to 0. The actions on writing to this bit are:

`0` No action

`1` Clear the EDSCR.{TXU, RXO, ERR} bits, and, if the core is in Debug state, the EDSCR.ITO bit, to 0.

RES0, [1:0]

RES0 Reserved.

The EDRCR can be accessed through the internal memory-mapped interface and the external debug interface, offset `0x090`.

Usage constraints

This register is accessible as follows:

Off	DLK	OSLK	SLK	Default
Error	Error	Error	WI	WO

Configurations

EDRCR is in the Core power domain.

Attributes

See *D3.1 Memory-mapped debug register summary* on page D3-602.

EDRCR is a 32-bit register.

Chapter D4
AArch32 PMU registers

This chapter describes the AArch32 PMU registers and shows examples of how to use them.

It contains the following sections:

D4.1 AArch32 PMU register summary

The PMU counters and their associated control registers are accessible in the AArch32 Execution state from the internal CP15 system register interface with `MCR` and `MRC` instructions for 32-bit registers and `MCRR` and `MRRC` for 64-bit registers.

The following table gives a summary of the Cortex-A75 PMU registers in the AArch32 Execution state. For those registers not described in this chapter, see the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

Table D4-1 PMU register summary in the AArch32 Execution state

CRn	Op1	CRm	Op2	Name	Type	Width	Reset	Description
c9	0	c12	0	PMCR	RW	32	`0x414A3000`	*D4.6 PMCR, Performance Monitors Control Register* on page D4-629
c9	0	c12	1	PMCNTENSET	RW	32	`0x00000000`	Performance Monitors Count Enable Set Register
c9	0	c12	2	PMCNTENCLR	RW	32	`0x00000000`	Performance Monitors Count Enable Clear Register
c9	0	c12	3	PMOVSR	RW	32	`0x00000000`	Performance Monitors Overflow Flag Status Register
c9	0	c12	4	PMSWINC	WO	32	UNK	Performance Monitors Software Increment Register
c9	0	c12	5	PMSELR	RW	32	UNK	Performance Monitors Event Counter Selection Register
c9	0	c12	6	PMCEID0	RO	32	`0xFBFF7F3F`	*D4.2 PMCEID0, Performance Monitors Common Event Identification Register 0* on page D4-622
c9	0	c12	7	PMCEID1	RO	32	`0x01F1AE7B`	*D4.3 PMCEID1, Performance Monitors Common Event Identification Register 1* on page D4-625
c9	0	c14	4	PMCEID2	RO	32	`0x00000000`	*D4.4 PMCEID2, Performance Monitors Common Event Identification Register 2* on page D4-627
c9	0	c14	5	PMCEID3	RO	32	`0x00000000`	*D4.5 PMCEID3, Performance Monitors Common Event Identification Register 3* on page D4-628
c9	0	c13	0	PMCCNTR[31:0]	RW	32	UNK	Performance Monitors Cycle Count Register
c9	3	c13	0	PMCCNTR[63:0]	RW	64	UNK	
c9	0	c13	1	PMXEVTYPER	RW	32	UNK	Performance Monitors Selected Event Type Register
c9	0	c13	2	PMXEVCNTR	RW	32	UNK	Performance Monitors Selected Event Count Register
c9	0	c14	0	PMUSERENR	RW	32	UNK	Performance Monitors User Enable Register
c9	0	c14	1	PMINTENSET	RW	32	`0x00000000`	Performance Monitors Interrupt Enable Set Register
c9	0	c14	2	PMINTENCLR	RW	32	`0x00000000`	Performance Monitors Interrupt Enable Clear Register
c9	0	c14	3	PMOVSSET	RW	32	`0x00000000`	Performance Monitor Overflow Flag Status Set Register
c14	0	c8	0	PMEVCNTR0	RW	32	UNK	Performance Monitor Event Count Registers
c14	0	c8	1	PMEVCNTR1	RW	32	UNK	
c14	0	c8	2	PMEVCNTR2	RW	32	UNK	
c14	0	c8	3	PMEVCNTR3	RW	32	UNK	
c14	0	c8	4	PMEVCNTR4	RW	32	UNK	
c14	0	c8	5	PMEVCNTR5	RW	32	UNK	

Table D4-1 PMU register summary in the AArch32 Execution state (continued)

CRn	Op1	CRm	Op2	Name	Type	Width	Reset	Description
c14	0	c12	0	PMEVTYPER0	RW	32	UNK	Performance Monitors Event Type Registers
c14	0	c12	1	PMEVTYPER1	RW	32	UNK	
c14	0	c12	2	PMEVTYPER2	RW	32	UNK	
c14	0	c12	3	PMEVTYPER3	RW	32	UNK	
c14	0	c12	4	PMEVTYPER4	RW	32	UNK	
c14	0	c12	5	PMEVTYPER5	RW	32	UNK	
c14	0	c15	7	PMCCFILTR	RW	32	UNK	Performance Monitors Cycle Count Filter Register

D4.2 PMCEID0, Performance Monitors Common Event Identification Register 0

The PMCEID0 defines which common architectural and common microarchitectural feature events are implemented.

Bit field descriptions

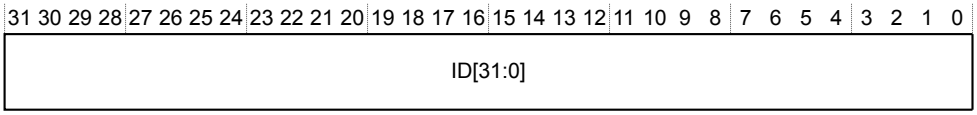

Figure D4-1 PMCEID0 bit assignments

ID[31:0], [31:0]

Common architectural and microarchitectural feature events that can be counted by the PMU event counters.

The following table shows the PMCEID0 bit assignments with event implemented or not implemented when the associated bit is set to 1 or 0. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile* for more information about these events.

Table D4-2 PMU events

Bit	Event number	Event mnemonic	Description
[31]	0x1F	L1D_CACHE_ALLOCATE	L1 Data cache allocate: 1 This event is implemented.
[30]	0x1E	CHAIN	Chain. For odd-numbered counters, counts once for each overflow of the preceding even-numbered counter. For even-numbered counters, does not count: 1 This event is implemented.
[29]	0x1D	BUS_CYCLES	Bus cycle: 1 This event is implemented.
[28]	0x1C	TTBR_WRITE_RETIRED	TTBR write, architecturally executed, condition check pass - write to translation table base: 1 This event is implemented.
[27]	0x1B	INST_SPEC	Instruction speculatively executed: 1 This event is implemented.
[26]	0x1A	MEMORY_ERROR	Local memory error: 0 This event is not implemented.
[25]	0x19	BUS_ACCESS	Bus access: 1 This event is implemented.
[24]	0x18	L2D_CACHE_WB	L2 Data cache Write-Back: 1 This event is implemented.

Table D4-2 PMU events (continued)

Bit	Event number	Event mnemonic	Description
[23]	0x17	L2D_CACHE_REFILL	L2 Data cache refill: 1 This event is implemented.
[22]	0x16	L2D_CACHE	L2 Data cache access: 1 This event is implemented.
[21]	0x15	L1D_CACHE_WB	L1 Data cache Write-Back: 1 This event is implemented.
[20]	0x14	L1I_CACHE	L1 Instruction cache access: 1 This event is implemented.
[19]	0x13	MEM_ACCESS	Data memory access: 1 This event is implemented.
[18]	0x12	BR_PRED	Predictable branch speculatively executed: 1 This event is implemented.
[17]	0x11	CPU_CYCLES	Cycle: 1 This event is implemented.
[16]	0x10	BR_MIS_PRED	Mispredicted or not predicted branch speculatively executed: 1 This event is implemented.
[15]	0x0F	UNALIGNED_LDST_RETIRED	Instruction architecturally executed, condition check pass - unaligned load or store: 0 This event is not implemented.
[14]	0x0E	BR_RETURN_RETIRED	Instruction architecturally executed, condition check pass - procedure return: 1 This event is implemented.
[13]	0x0D	BR_IMMED_RETIRED	Instruction architecturally executed - immediate branch: 1 This event is implemented.
[12]	0x0C	PC_WRITE_RETIRED	Instruction architecturally executed, condition check pass - software change of the PC: 1 This event is implemented.
[11]	0x0B	CID_WRITE_RETIRED	Instruction architecturally executed, condition check pass - write to CONTEXTIDR: 1 This event is implemented.
[10]	0x0A	EXC_RETURN	Instruction architecturally executed, condition check pass - exception return: 1 This event is implemented.

Table D4-2 PMU events (continued)

Bit	Event number	Event mnemonic	Description
[9]	0x09	EXC_TAKEN	Exception taken: 1 This event is implemented.
[8]	0x08	INST_RETIRED	Instruction architecturally executed: 1 This event is implemented.
[7]	0x07	ST_RETIRED	Instruction architecturally executed, condition check pass - store: 0 This event is not implemented.
[6]	0x06	LD_RETIRED	Instruction architecturally executed, condition check pass - load: 0 This event is not implemented.
[5]	0x05	L1D_TLB_REFILL	L1 Data TLB refill: 1 This event is implemented.
[4]	0x04	L1D_CACHE	L1 Data cache access: 1 This event is implemented.
[3]	0x03	L1D_CACHE_REFILL	L1 Data cache refill: 1 This event is implemented.
[2]	0x02	L1I_TLB_REFILL	L1 Instruction TLB refill: 1 This event is implemented.
[1]	0x01	L1I_CACHE_REFILL	L1 Instruction cache refill: 1 This event is implemented.
[0]	0x00	SW_INCR	Instruction architecturally executed, condition check pass - software increment: 1 This event is implemented.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

D4.3 PMCEID1, Performance Monitors Common Event Identification Register 1

The PMCEID1 defines which common architectural and common microarchitectural feature events are implemented.

Bit field descriptions

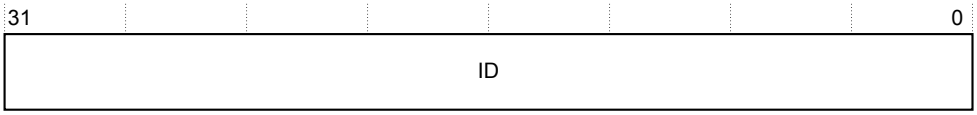

Figure D4-2 PMCEID1 bit assignments

ID, [31:0]

Common architectural and microarchitectural feature events that the PMU event counters can count.

For each bit described in the following table, the event is implemented if the bit is set to 1, or not implemented if the bit is set to 0.

Table D4-3 PMU common events

Bit	Event number	Event mnemonic	Description
[31:25]	0x3F-0x39	-	0 Events are not implemented.
[24]	0x38	REMOTE_ACCESS	Access to another socket in a multi-socket system. 1 This event is implemented.
[23]	0x37	LL_CACHE_RD_MISS	Last level cache access, read. 1 This event is implemented.
[22]	0x36	LL_CACHE_RD	Last level cache access, read. 1 This event is implemented.
[21]	0x35	ITLB_WALK	Last level cache access, read. 1 This event is implemented.
[20]	0x34	DTLB_WALK	Access to instruction TLB that caused a translation table walk. 1 This event is implemented.
[19:17]	0x33-0x31	-	0 Events are not implemented.
[16]	0x30	L2I_TLB	Attributable Level 2 instruction TLB access. 1 This event is implemented.
[15]	0x2F	L2D_TLB	Attributable Level 2 data or unified TLB access. 1 This event is implemented.
[14]	0x2E	-	0 This event is not implemented.

Table D4-3 PMU common events (continued)

Bit	Event number	Event mnemonic	Description
[13]	0x2D	L2D_TLB_REFILL	Attributable Level 2 data or unified TLB refill. 1 This event is implemented.
[12]	0x2C	-	0 This event is not implemented.
[11]	0x2B	L3D_CACHE	Attributable Level 3 data or unified cache access. 1 This event is implemented.
[10]	0x2A	L3D_CACHE_REFILL	Attributable Level 3 data or unified cache refill. 1 This event is implemented.
[9]	0x29	L3D_CACHE_ALLOCATE	Attributable Level 3 data or unified cache allocation without refill. 1 This event is implemented.
[8]	0x28	-	0 This event is not implemented.
[7]	0x27	-	0 This event is not implemented.
[6]	0x26	L1I_TLB	Level 1 instruction TLB access. 1 This event is implemented.
[5]	0x25	L1D_TLB	Level 1 data or unified TLB access. 1 This event is implemented.
[4]	0x24	STALL_BACKEND	No operation issued due to backend. 1 This event is implemented.
[3]	0x23	STALL_FRONTEND	No operation issued due to frontend. 1 This event is implemented.
[2]	0x22	-	0 This event is not implemented.
[1]	0x21	BR_RETIRED	Instruction architecturally executed, branch. 1 This event is implemented.
[0]	0x20	L2D_CACHE_ALLOCATE	Level 2 data cache allocation without refill. 1 This event is implemented.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

D4.4 PMCEID2, Performance Monitors Common Event Identification Register 2

The PMCEID2 defines which architectural and common microarchitectural feature events in the range `0x4000-0x401F` are implemented.

Bit field descriptions

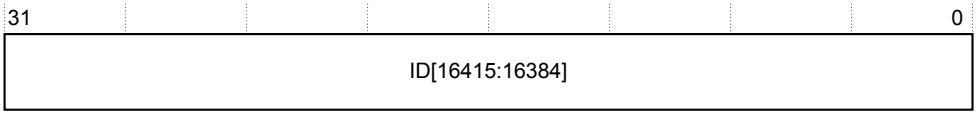

Figure D4-3 PMCEID2 bit assignments

ID[16415:16384], bits [31:0]

PMCEID2[31:0] maps to common events `0x4000-0x401F`. The value is:

0 These events are not implemented.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

D4.5 PMCEID3, Performance Monitors Common Event Identification Register 3

The PMCEID3 defines which architectural and common microarchitectural feature events in the range `0x4020-0x403F` are implemented.

Bit field descriptions

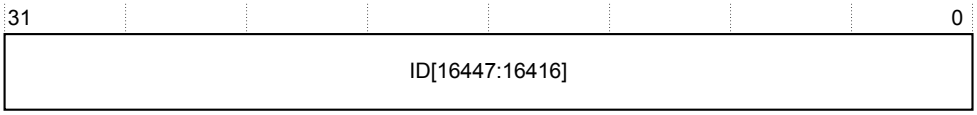

Figure D4-4 PMCEID3 bit assignments

ID[16447:16416], bits [31:0]

PMCEID2[31:0] maps to common events `0x4020-0x403F`. The value is:

0 These events are not implemented.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

D4.6 PMCR, Performance Monitors Control Register

The PMCR provides details of the Performance Monitors implementation, including the number of counters implemented, and configures and controls the counters.

Bit field descriptions

PMCR is a 32-bit register, and is part of the Performance Monitors registers functional group.

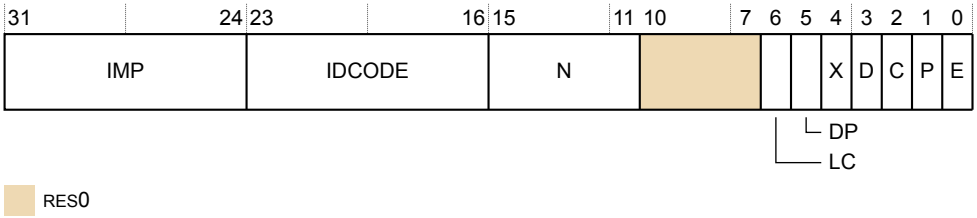

Figure D4-5 PMCR bit assignments

IMP, [31:24]
Indicates the implementer code. The value is:

`0x41` ASCII character 'A' - implementer is Arm Limited.

IDCODE, [23:16]
Identification code. The value is:

`0x4A` Cortex-A75 core.

N, [15:11]
Identifies the number of event counters implemented.

`0b00110` The core implements six event counters.

RES0, [10:7]

RES0 Reserved.

LC, [6]
Long cycle count enable. Determines which PMCCNTR bit generates an overflow recorded in PMOVSR[31]. The overflow event is generated on a 32-bit or 64-bit boundary. The possible values are:

`0b0` Overflow event is generated on a 32-bit boundary, when an increment changes PMCCNTR[31] from 1 to 0. This is the reset value.

`0b1` Overflow event is generated on a 64-bit boundary, when an increment changes PMCCNTR[63] from 1 to 0.

Arm deprecates use of PMCR.LC = `0b0`.

DP, [5]
Disable cycle counter CCNT when event counting is prohibited. The possible values are:

`0b0` Cycle counter operates regardless of the non-invasive debug authentication settings. This is the reset value.

`0b1` Cycle counter is disabled if non-invasive debug is not permitted and enabled.

X, [4]
Export enable. This bit permits events to be exported to another debug device, such as a trace macrocell, over an event bus. The possible values are:

`0b0` Export of events is disabled. This is the reset value.

`0b1` Export of events is enabled.

No events are exported when counting is prohibited.

This field does not affect the generation of Performance Monitors overflow interrupt requests or signaling to a cross-trigger interface (CTI) that can be implemented as signals exported from the PE.

When this register has an architecturally defined reset value, if this field is implemented as an RW field, it resets to 0.

D, [3]

Clock divider. The possible values are:

`0b0` When enabled, counter CCNT counts every clock cycle. This is the reset value.

`0b1` When enabled, counter CCNT counts once every 64 clock cycles.

Arm deprecates use of PMCR.D = `0b1`.

C, [2]

Cycle counter reset. This bit is WO. The effects of writing to this bit are:

`0b0` No action. This is the reset value.

`0b1` Reset PMCCNTR to zero.

This bit is always RAZ.

Resetting PMCCNTR does not clear the PMCCNTR overflow bit to 0. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile* for more information.

P, [1]

Event counter reset. This bit is WO. The effects of writing to this bit are:

`0b0` No action. This is the reset value.

`0b1` Reset all event counters accessible in the current EL, not including PMCCNTR, to zero.

This bit is always RAZ.

In Non-secure EL0 and EL1, a write of 1 to this bit does not reset event counters that HDCR.HPMN or MDCR_EL2.HPMN reserves for EL2 use.

In EL2 and EL3, a write of 1 to this bit resets all the event counters.

Resetting the event counters does not clear any overflow bits to 0.

E, [0]

Enable. The possible values are:

`0b0` All counters that are accessible at Non-secure EL1, including PMCCNTR, are disabled. This is the reset value.

`0b1` When this register has an architecturally defined reset value, this field resets to 0.

This bit is RW.

This bit does not affect the operation of event counters that HDCR.HPMN or MDCR_EL2.HPMN reserves for EL2 use.

When this register has an architecturally defined reset value, this field resets to 0.

Configurations

AArch32 System register PMCR is architecturally mapped to AArch64 System register PMCR_EL0. See *D5.4 PMCR_EL0, Performance Monitors Control Register, EL0* on page D5-641.

AArch32 System register PMCR bits [6:0] are architecturally mapped to External register PMCR_EL0[6:0].

There is one instance of this register that is used in both Secure and Non-secure states.

This register is in the Warm reset domain. Some or all RW fields of this register have defined reset values. On a Warm or Cold reset these apply only if the PE resets into an Exception level that is using AArch32. Otherwise, on a Warm or Cold reset RW fields in this register reset to architecturally UNKNOWN values.

Chapter D5
AArch64 PMU registers

This chapter describes the AArch64 PMU registers and shows examples of how to use them.

It contains the following sections:

D5.1 AArch64 PMU register summary

The PMU counters and their associated control registers are accessible in the AArch64 Execution state with `MRS` and `MSR` instructions.

The following table gives a summary of the Cortex-A75 PMU registers in the AArch64 Execution state. For those registers not described in this chapter, see the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

Table D5-1 PMU register summary in the AArch64 Execution state

Name	Type	Width	Reset	Description
PMCR_EL0	RW	32	`0x414A3000`	*D5.4 PMCR_EL0, Performance Monitors Control Register, EL0* on page D5-641
PMCNTENSET_EL0	RW	32	UNK	Performance Monitors Count Enable Set Register
PMCNTENCLR_EL0	RW	32	UNK	Performance Monitors Count Enable Clear Register
PMOVSCLR_EL0	RW	32	UNK	Performance Monitors Overflow Flag Status Register
PMSWINC_EL0	WO	32	UNK	Performance Monitors Software Increment Register
PMSELR_EL0	RW	32	UNK	Performance Monitors Event Counter Selection Register
PMCEID0_EL0	RO	64	`0xFBFF7F3F`	*D5.2 PMCEID0_EL0, Performance Monitors Common Event Identification Register 0_EL0* on page D5-636
PMCEID1_EL0	RO	64	`0x01F1AE7B`	*D5.3 PMCEID1_EL0, Performance Monitors Common Event Identification Register 1, EL0* on page D5-639
PMCCNTR_EL0	RW	64	UNK	Performance Monitors Cycle Count Register
PMXEVTYPER_EL0	RW	32	UNK	Performance Monitors Selected Event Type and Filter Register
PMCCFILTR_EL0	RW	32	UNK	Performance Monitors Cycle Count Filter Register

Table D5-1 PMU register summary in the AArch64 Execution state (continued)

Name	Type	Width	Reset	Description
PMXEVCNTR_EL0	RW	32	UNK	Performance Monitors Selected Event Count Register
PMUSERENR_EL0	RW	32	UNK	Performance Monitors User Enable Register
PMINTENSET_EL1	RW	32	UNK	Performance Monitors Interrupt Enable Set Register
PMINTENCLR_EL1	RW	32	UNK	Performance Monitors Interrupt Enable Clear Register
PMOVSSET_EL0	RW	32	UNK	Performance Monitors Overflow Flag Status Set Register
PMEVCNTR0_EL0	RW	32	UNK	Performance Monitors Event Count Registers
PMEVCNTR1_EL0	RW	32	UNK	
PMEVCNTR2_EL0	RW	32	UNK	
PMEVCNTR3_EL0	RW	32	UNK	
PMEVCNTR4_EL0	RW	32	UNK	
PMEVCNTR5_EL0	RW	32	UNK	
PMEVTYPER0_EL0	RW	32	UNK	Performance Monitors Event Type Registers
PMEVTYPER1_EL0	RW	32	UNK	
PMEVTYPER2_EL0	RW	32	UNK	
PMEVTYPER3_EL0	RW	32	UNK	
PMEVTYPER4_EL0	RW	32	UNK	
PMEVTYPER5_EL0	RW	32	UNK	
PMCCFILTR_EL0	RW	32	UNK	Performance Monitors Cycle Count Filter Register

D5.2 PMCEID0_EL0, Performance Monitors Common Event Identification Register 0_EL0

The PMCEID0_EL0 defines which common architectural and common microarchitectural feature events are implemented.

Bit field descriptions

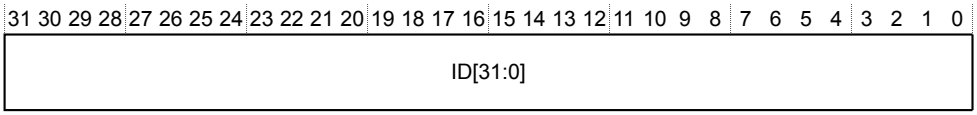

Figure D5-1 PMCEID0_EL0 bit assignments

ID[31:0], [31:0]

Common architectural and microarchitectural feature events that can be counted by the PMU event counters.

For each bit described in the following table, the event is implemented if the bit is set to 1, or not implemented if the bit is set to 0.

Table D5-2 PMU common events

Bit	Event number	Event mnemonic	Description
[31]	0x1F	L1D_CACHE_ALLOCATE	L1 Data cache allocate: 1 This event is implemented.
[30]	0x1E	CHAIN	Chain. For odd-numbered counters, counts once for each overflow of the preceding even-numbered counter. For even-numbered counters, does not count: 1 This event is implemented.
[29]	0x1D	BUS_CYCLES	Bus cycle: 1 This event is implemented.
[28]	0x1C	TTBR_WRITE_RETIRED	TTBR write, architecturally executed, condition check pass - write to translation table base: 1 This event is implemented.
[27]	0x1B	INST_SPEC	Instruction speculatively executed: 1 This event is implemented.
[26]	0x1A	MEMORY_ERROR	Local memory error: 0 This event is not implemented.
[25]	0x19	BUS_ACCESS	Bus access: 1 This event is implemented.
[24]	0x18	L2D_CACHE_WB	L2 Data cache Write-Back: 1 This event is implemented.

Table D5-2 PMU common events (continued)

Bit	Event number	Event mnemonic	Description
[23]	0x17	L2D_CACHE_REFILL	L2 Data cache refill: 1 This event is implemented.
[22]	0x16	L2D_CACHE	L2 Data cache access: 1 This event is implemented.
[21]	0x15	L1D_CACHE_WB	L1 Data cache Write-Back: 1 This event is implemented.
[20]	0x14	L1I_CACHE	L1 Instruction cache access: 1 This event is implemented.
[19]	0x13	MEM_ACCESS	Data memory access: 1 This event is implemented.
[18]	0x12	BR_PRED	Predictable branch speculatively executed: 1 This event is implemented.
[17]	0x11	CPU_CYCLES	Cycle: 1 This event is implemented.
[16]	0x10	BR_MIS_PRED	Mispredicted or not predicted branch speculatively executed: 1 This event is implemented.
[15]	0x0F	UNALIGNED_LDST_RETIRED	Instruction architecturally executed, condition check pass - unaligned load or store: 0 This event is not implemented.
[14]	0x0E	BR_RETURN_RETIRED	Instruction architecturally executed, condition check pass - procedure return: 1 This event is implemented.
[13]	0x0D	BR_IMMED_RETIRED	Instruction architecturally executed - immediate branch: 1 This event is implemented.
[12]	0x0C	PC_WRITE_RETIRED	Instruction architecturally executed, condition check pass - software change of the PC: 1 This event is implemented.
[11]	0x0B	CID_WRITE_RETIRED	Instruction architecturally executed, condition check pass - write to CONTEXTIDR: 1 This event is implemented.
[10]	0x0A	EXC_RETURN	Instruction architecturally executed, condition check pass - exception return: 1 This event is implemented.

Table D5-2 PMU common events (continued)

Bit	Event number	Event mnemonic	Description
[9]	0x09	EXC_TAKEN	Exception taken: 1 This event is implemented.
[8]	0x08	INST_RETIRED	Instruction architecturally executed: 1 This event is implemented.
[7]	0x07	ST_RETIRED	Instruction architecturally executed, condition check pass - store: 0 This event is not implemented.
[6]	0x06	LD_RETIRED	Instruction architecturally executed, condition check pass - load: 0 This event is not implemented.
[5]	0x05	L1D_TLB_REFILL	L1 Data TLB refill: 1 This event is implemented.
[4]	0x04	L1D_CACHE	L1 Data cache access: 1 This event is implemented.
[3]	0x03	L1D_CACHE_REFILL	L1 Data cache refill: 1 This event is implemented.
[2]	0x02	L1I_TLB_REFILL	L1 Instruction TLB refill: 1 This event is implemented.
[1]	0x01	L1I_CACHE_REFILL	L1 Instruction cache refill: 1 This event is implemented.
[0]	0x00	SW_INCR	Instruction architecturally executed, condition check pass - software increment: 1 This event is implemented.

D5.3 PMCEID1_EL0, Performance Monitors Common Event Identification Register 1, EL0

The PMCEID1_EL0 defines which common architectural and common microarchitectural feature events are implemented.

Bit field descriptions

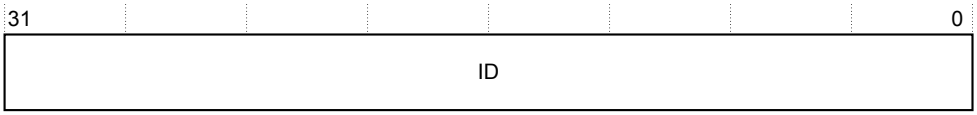

Figure D5-2 PMCEID1_EL0 bit assignments

ID[63:32], [31:0]

Common architectural and microarchitectural feature events that can be counted by the PMU event counters.

For each bit described in the following table, the event is implemented if the bit is set to 1, or not implemented if the bit is set to 0.

Table D5-3 PMU common events

Bit	Event number	Event mnemonic	Description
[31:25]	0x3F-0x39	-	0 Events are not implemented.
[24]	0x38	REMOTE_ACCESS	Access to another socket in a multi-socket system. 1 This event is implemented.
[23]	0x37	LL_CACHE_RD_MISS	Last level cache access, read. 1 This event is implemented.
[22]	0x36	LL_CACHE_RD	Last level cache access, read. 1 This event is implemented.
[21]	0x35	ITLB_WALK	Last level cache access, read. 1 This event is implemented.
[20]	0x34	DTLB_WALK	Access to instruction TLB that caused a translation table walk. 1 This event is implemented.
[19:17]	0x33-0x31	-	0 Events are not implemented.
[16]	0x30	L2I_TLB	Attributable Level 2 instruction TLB access. 1 This event is implemented.
[15]	0x2F	L2D_TLB	Attributable Level 2 data or unified TLB access. 1 This event is implemented.
[14]	0x2E	-	0 This event is not implemented.

Table D5-3 PMU common events (continued)

Bit	Event number	Event mnemonic	Description
[13]	0x2D	L2D_TLB_REFILL	Attributable Level 2 data or unified TLB refill. 1 This event is implemented.
[12]	0x2C	-	0 This event is not implemented.
[11]	0x2B	L3D_CACHE	Attributable Level 3 data or unified cache access. 1 This event is implemented.
[10]	0x2A	L3D_CACHE_REFILL	Attributable Level 3 data or unified cache refill. 1 This event is implemented.
[9]	0x29	L3D_CACHE_ALLOCATE	Attributable Level 3 data or unified cache allocation without refill. 1 This event is implemented.
[8]	0x28	-	0 This event is not implemented.
[7]	0x27	-	0 This event is not implemented.
[6]	0x26	L1I_TLB	Level 1 instruction TLB access. 1 This event is implemented.
[5]	0x25	L1D_TLB	Level 1 data or unified TLB access. 1 This event is implemented.
[4]	0x24	STALL_BACKEND	No operation issued due to backend. 1 This event is implemented.
[3]	0x23	STALL_FRONTEND	No operation issued due to frontend. 1 This event is implemented.
[2]	0x22	-	0 This event is not implemented.
[1]	0x21	BR_RETIRED	Instruction architecturally executed, branch. 1 This event is implemented.
[0]	0x20	L2D_CACHE_ALLOCATE	Level 2 data cache allocation without refill. 1 This event is implemented.

D5.4 PMCR_EL0, Performance Monitors Control Register, EL0

The PMCR_EL0 provides details of the Performance Monitors implementation, including the number of counters implemented, and configures and controls the counters.

Bit field descriptions

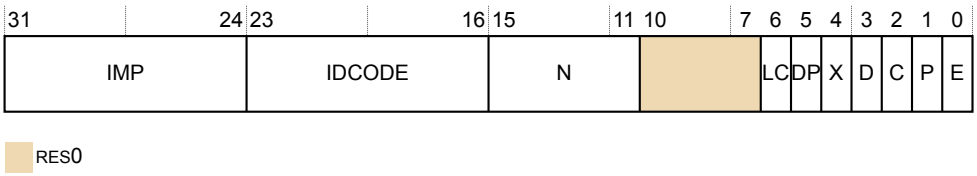

Figure D5-3 PMCR_EL0 bit assignments

IMP, [31:24]
Implementer code:

`0x41` Arm.

This is a read-only field.

IDCODE, [23:16]
Identification code:

`0x4A` Cortex-A75.

This is a read-only field.

N, [15:11]
Number of event counters.

`0b00110` Six counters.

RES0, [10:7]

RES0 Reserved.

LC, [6]
Long cycle count enable. Determines which PMCCNTR_EL0 bit generates an overflow recorded in PMOVSR[31]. The possible values are:

`0` Overflow on increment that changes PMCCNTR_EL0[31] from 1 to 0.

`1` Overflow on increment that changes PMCCNTR_EL0[63] from 1 to 0.

DP, [5]
Disable cycle counter, PMCCNTR_EL0 when event counting is prohibited:

`0` Cycle counter operates regardless of the non-invasive debug authentication settings. This is the reset value.

`1` Cycle counter is disabled if non-invasive debug is not permitted and enabled.

This bit is read/write.

X, [4]
Export enable. This bit permits events to be exported to another debug device, such as a trace macrocell, over an event bus:

`0` Export of events is disabled. This is the reset value.

`1` Export of events is enabled.

This bit is read/write and does not affect the generation of Performance Monitors interrupts on the **nPMUIRQ** pin.

D, [3]

Clock divider:

0 When enabled, PMCCNTR_EL0 counts every clock cycle. This is the reset value.

1 When enabled, PMCCNTR_EL0 counts every 64 clock cycles.

This bit is read/write.

C, [2]

Clock counter reset. This bit is WO. The effects of writing to this bit are:

0 No action. This is the reset value.

1 Reset PMCCNTR_EL0 to 0.

This bit is always RAZ.

Resetting PMCCNTR_EL0 does not clear the PMCCNTR_EL0 overflow bit to 0. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile* for more information.

P, [1]

Event counter reset. This bit is WO. The effects of writing to this bit are:

0 No action. This is the reset value.

1 Reset all event counters, not including PMCCNTR_EL0, to zero.

This bit is always RAZ.

In Non-secure EL0 and EL1, a write of 1 to this bit does not reset event counters that MDCR_EL2.HPMN reserves for EL2 use.

In EL2 and EL3, a write of 1 to this bit resets all the event counters.

Resetting the event counters does not clear any overflow bits to 0.

E, [0]

Enable. The possible values of this bit are:

0 All counters, including PMCCNTR_EL0, are disabled. This is the reset value.

1 All counters are enabled.

This bit is RW.

In Non-secure EL0 and EL1, this bit does not affect the operation of event counters that MDCR_EL2.HPMN reserves for EL2 use.

On Warm reset, the field resets to 0.

Configurations

AArch64 System register PMCR_EL0 is architecturally mapped to AArch32 System register PMCR.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

Chapter D6
Memory-mapped PMU registers

This chapter describes the memory-mapped PMU registers and shows examples of how to use them.

It contains the following sections:

D6.1 Memory-mapped PMU register summary

There are PMU registers that are accessible through the external debug interface.

These registers are listed in the following table. For those registers not described in this chapter, see the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

Table D6-1 Memory-mapped PMU register summary

Offset	Name	Type	Description
0x000	PMEVCNTR0_EL0	RW	Performance Monitor Event Count Register 0
0x004	-	-	Reserved
0x008	PMEVCNTR1_EL0	RW	Performance Monitor Event Count Register 1
0x00C	-	-	Reserved
0x010	PMEVCNTR2_EL0	RW	Performance Monitor Event Count Register 2
0x014	-	-	Reserved
0x018	PMEVCNTR3_EL0	RW	Performance Monitor Event Count Register 3
0x01C	-	-	Reserved
0x020	PMEVCNTR4_EL0	RW	Performance Monitor Event Count Register 4
0x024	-	-	Reserved
0x028	PMEVCNTR5_EL0	RW	Performance Monitor Event Count Register 5
0x02C-0xF4	-	-	Reserved
0x0F8	PMCCNTR_EL0[31:0]	RW	Performance Monitor Cycle Count Register
0x0FC	PMCCNTR_EL0[63:32]	RW	
0x200	PMPCSR[31:0]	RO	Program Counter Sample Register
0x204	PMPCSR[63:32]		
0x208	PMCID1SR	RO	CONTEXTIDR_EL1 Sample Register
0x20C	PMVIDSR	RO	VMID Sample Register
0x220	PMPCSR[31:0]	RO	Program Counter Sample Register (alias)
0x224	PMPCSR[63:32]		
0x228	PMCID1SR	RO	CONTEXTIDR_EL1 Sample Register (alias)
0x22C	PMCID2SR	RO	CONTEXTIDR_EL2 Sample Register
0x100-0x3FC	-	-	Reserved
0x418-0x478	-	-	Reserved
0x47C	PMCCFILTR_EL0	RW	Performance Monitor Cycle Count Filter Register

Table D6-1 Memory-mapped PMU register summary (continued)

Offset	Name	Type	Description
0x600	PMPCSSR_LO	RO	*D9.2 PMPCSSR, Snapshot Program Counter Sample Register* on page D9-681
0x604	PMPCSSR_HI	RO	
0x608	PMCIDSSR	RO	*D9.3 PMCIDSSR, Snapshot CONTEXTIDR_EL1 Sample Register* on page D9-682
0x60C	PMCID2SSR	RO	*D9.4 PMCID2SSR, Snapshot CONTEXTIDR_EL2 Sample Register* on page D9-683
0x610	PMSSSR	RO	*D9.5 PMSSSR, PMU Snapshot Status Register* on page D9-684
0x614	PMOVSSR	RO	*D9.6 PMOVSSR, PMU Overflow Status Snapshot Register* on page D9-685
0x618	PMCCNTSR_LO	RO	*D9.7 PMCCNTSR, PMU Cycle Counter Snapshot Register* on page D9-686
0x61C	PMCCNTSR_HI	RO	
0x620+ 4×n	PMEVCNTSR<n>	RO	*D9.8 PMEVCNTSR 0-5, PMU Cycle Counter Snapshot Registers* on page D9-687
0x6F0	PMSSCR	WO	*D9.9 PMSSCR, PMU Snapshot Capture Register* on page D9-688
0xC00	PMCNTENSET_EL0	RW	Performance Monitor Count Enable Set Register
0xC04-0xC1C	-	-	Reserved
0xC20	PMCNTENCLR_EL0	RW	Performance Monitor Count Enable Clear Register
0xC24-0xC3C	-	-	Reserved
0xC40	PMINTENSET_EL1	RW	Performance Monitor Interrupt Enable Set Register
0xC44-0xC5C	-	-	Reserved
0xC60	PMINTENCLR_EL1	RW	Performance Monitor Interrupt Enable Clear Register
0xC64-0xC7C	-	-	Reserved
0xC80	PMOVSCLR_EL0	RW	Performance Monitor Overflow Flag Status Register
0xC84-0xC9C	-	-	Reserved
0xCA0	PMSWINC_EL0	WO	Performance Monitor Software Increment Register
0xCA4-0xCBC	-	-	Reserved
0xCC0	PMOVSSET_EL0	RW	Performance Monitor Overflow Flag Status Set Register
0xCC4-0xDFC	-	-	Reserved

Table D6-1 Memory-mapped PMU register summary (continued)

Offset	Name	Type	Description
0xE00	PMCFGR	RO	*D6.2 PMCFGR, Performance Monitors Configuration Register* on page D6-648
0xE04	PMCR_EL0	RW	Performance Monitors Control Register. This register is distinct from the PMCR_EL0 system register. It does not have the same value.
0xE08-0xE1C	-	-	Reserved
0xE20	PMCEID0	RO	*D5.2 PMCEID0_EL0, Performance Monitors Common Event Identification Register 0_EL0* on page D5-636
0xE24	PMCEID1	RO	*D5.3 PMCEID1_EL0, Performance Monitors Common Event Identification Register 1, EL0* on page D5-639
0xE28	PMCEID2	RO	*D4.4 PMCEID2, Performance Monitors Common Event Identification Register 2* on page D4-627
0xE2C	PMCEID3	RO	*D4.5 PMCEID3, Performance Monitors Common Event Identification Register 3* on page D4-628
0xFA4	-	-	Reserved
0xFA8	PMDEVAFF0	RO	*B2.84 MPIDR_EL1, Multiprocessor Affinity Register, EL1* on page B2-409
0xFAC	PMDEVAFF1	RO	*B2.84 MPIDR_EL1, Multiprocessor Affinity Register, EL1* on page B2-409
0xFB8	PMAUTHSTATUS	RO	Performance Monitor Authentication Status Register
0xFBC	PMDEVARCH	RO	Performance Monitor Device Architecture Register
0xFC0-0xFC8	-	-	Reserved
0xFCC	PMDEVTYPE	RO	Performance Monitor Device Type Register
0xFD0	PMPIDR4	RO	*D6.11 PMPIDR4, Performance Monitors Peripheral Identification Register 4* on page D6-657
0xFD4	PMPIDR5	RO	*D6.12 PMPIDRn, Performance Monitors Peripheral Identification Register 5-7* on page D6-658
0xFD8	PMPIDR6	RO	
0xFDC	PMPIDR7	RO	
0xFE0	PMPIDR0	RO	*D6.7 PMPIDR0, Performance Monitors Peripheral Identification Register 0* on page D6-653
0xFE4	PMPIDR1	RO	*D6.8 PMPIDR1, Performance Monitors Peripheral Identification Register 1* on page D6-654

Table D6-1 Memory-mapped PMU register summary (continued)

Offset	Name	Type	Description
0xFE8	PMPIDR2	RO	*D6.9 PMPIDR2, Performance Monitors Peripheral Identification Register 2* on page D6-655
0xFEC	PMPIDR3	RO	*D6.10 PMPIDR3, Performance Monitors Peripheral Identification Register 3* on page D6-656
0xFF0	PMCIDR0	RO	*D6.3 PMCIDR0, Performance Monitors Component Identification Register 0* on page D6-649
0xFF4	PMCIDR1	RO	*D6.4 PMCIDR1, Performance Monitors Component Identification Register 1* on page D6-650
0xFF8	PMCIDR2	RO	*D6.5 PMCIDR2, Performance Monitors Component Identification Register 2* on page D6-651
0xFFC	PMCIDR3	RO	*D6.6 PMCIDR3, Performance Monitors Component Identification Register 3* on page D6-652

D6.2 PMCFGR, Performance Monitors Configuration Register

The PMCFGR contains PMU specific configuration data.

Bit field descriptions

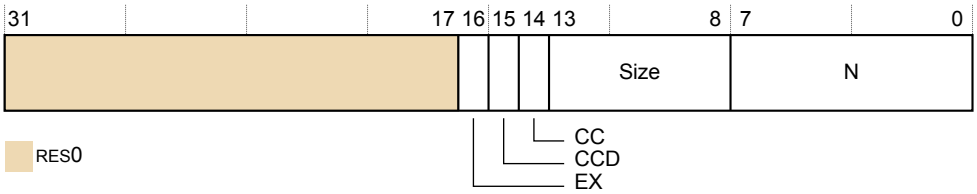

Figure D6-1 PMCFGR bit assignments

RES0, [31:17]

RES0 Reserved.

EX, [16]

Export supported. The value is:

`1` Export is supported. PMCR_EL0.EX is read/write.

CCD, [15]

Cycle counter has pre-scale. The value is:

`1` PMCR_EL0.D is read/write.

CC, [14]

Dedicated cycle counter supported. The value is:

`1` Dedicated cycle counter is supported.

Size, [13:8]

Counter size. The value is:

`0b111111` 64-bit counters.

N, [7:0]

Number of event counters. The value is:

`0x06` Six counters.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The PMCFGR can be accessed through the external debug interface, offset `0xE00`.

D6.3 PMCIDR0, Performance Monitors Component Identification Register 0

The PMCIDR0 provides information to identify a Performance Monitor component.

Bit field descriptions

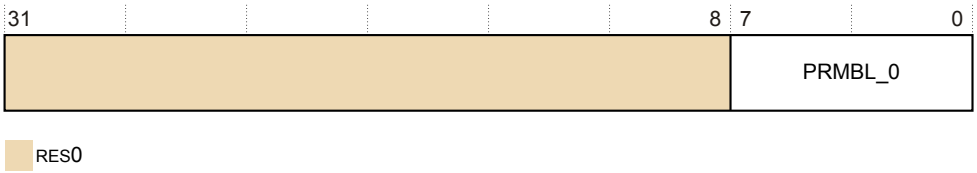

Figure D6-2 PMCIDR0 bit assignments

RES0, [31:8]

RES0 Reserved.

PRMBL_0, [7:0]

`0x0D` Preamble byte 0.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The PMCIDR0 can be accessed through the external debug interface, offset `0xFF0`.

D6.4 PMCIDR1, Performance Monitors Component Identification Register 1

The PMCIDR1 provides information to identify a Performance Monitor component.

Bit field descriptions

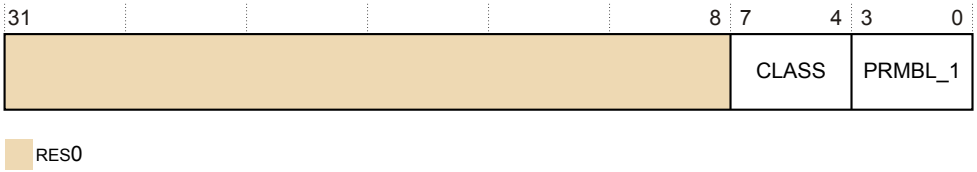

Figure D6-3 PMCIDR1 bit assignments

RES0, [31:8]

RES0 Reserved.

CLASS, [7:4]

`0x9` Debug component.

PRMBL_1, [3:0]

`0x0` Preamble byte 1.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The PMCIDR1 can be accessed through the external debug interface, offset `0xFF4`.

D6.5 PMCIDR2, Performance Monitors Component Identification Register 2

The PMCIDR2 provides information to identify a Performance Monitor component.

Bit field descriptions

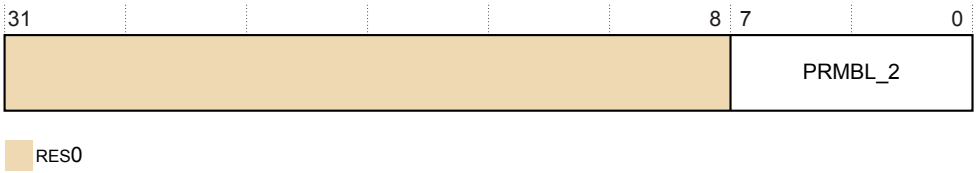

Figure D6-4 PMCIDR2 bit assignments

RES0, [31:8]

RES0 Reserved.

PRMBL_2, [7:0]

`0x05` Preamble byte 2.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The PMCIDR2 can be accessed through the external debug interface, offset `0xFF8`.

D6.6 PMCIDR3, Performance Monitors Component Identification Register 3

The PMCIDR3 provides information to identify a Performance Monitor component.

Bit field descriptions

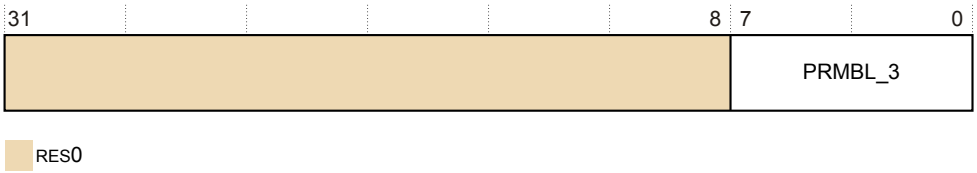

Figure D6-5 PMCIDR3 bit assignments

RES0, [31:8]

RES0 Reserved.

PRMBL_3, [7:0]

`0xB1` Preamble byte 3.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The PMCIDR3 can be accessed through the external debug interface, offset `0xFFC`.

D6.7 PMPIDR0, Performance Monitors Peripheral Identification Register 0

The PMPIDR0 provides information to identify a Performance Monitor component.

Bit field descriptions

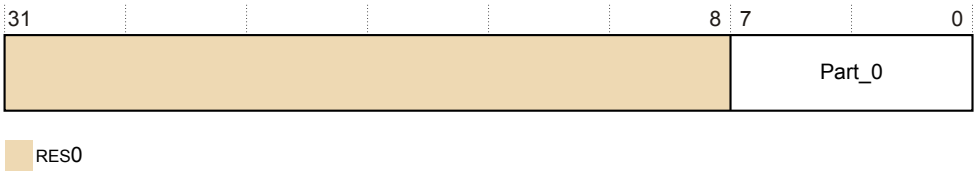

Figure D6-6 PMPIDR0 bit assignments

RES0, [31:8]

RES0 Reserved.

Part_0, [7:0]

`0x0A` Least significant byte of the performance monitor part number.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The PMPIDR0 can be accessed through the external debug interface, offset `0xFE0`.

D6.8 PMPIDR1, Performance Monitors Peripheral Identification Register 1

The PMPIDR1 provides information to identify a Performance Monitor component.

Bit field descriptions

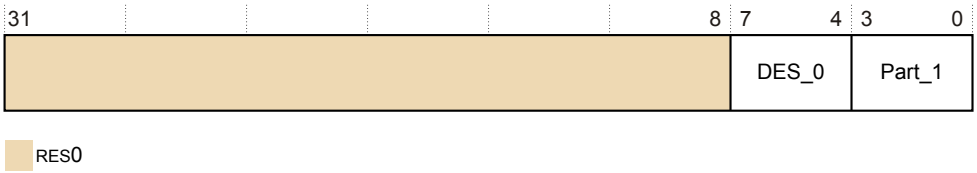

Figure D6-7 PMPIDR1 bit assignments

RES0, [31:8]

RES0 Reserved.

DES_0, [7:4]

`0xB` Arm Limited. This is the least significant nibble of JEP106 ID code.

Part_1, [3:0]

`0xD` Most significant nibble of the performance monitor part number.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The PMPIDR1 can be accessed through the external debug interface, offset `0xFE4`.

D6.9 PMPIDR2, Performance Monitors Peripheral Identification Register 2

The PMPIDR2 provides information to identify a Performance Monitor component.

Bit field descriptions

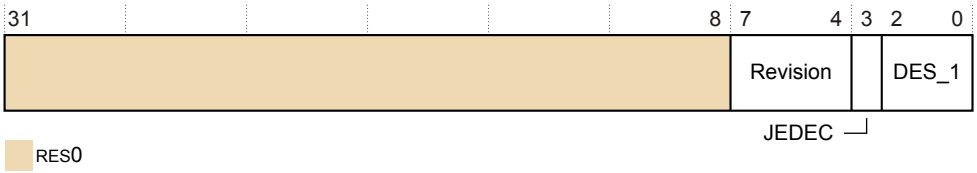

Figure D6-8 PMPIDR2 bit assignments

RES0, [31:8]

RES0 Reserved.

Revision, [7:4]

`0x2` r2p1.

JEDEC, [3]

`0b1` RAO. Indicates a JEP106 identity code is used.

DES_1, [2:0]

`0b011` Arm Limited. This is the most significant nibble of JEP106 ID code.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The PMPIDR2 can be accessed through the external debug interface, offset `0xFE8`.

D6.10 PMPIDR3, Performance Monitors Peripheral Identification Register 3

The PMPIDR3 provides information to identify a Performance Monitor component.

Bit field descriptions

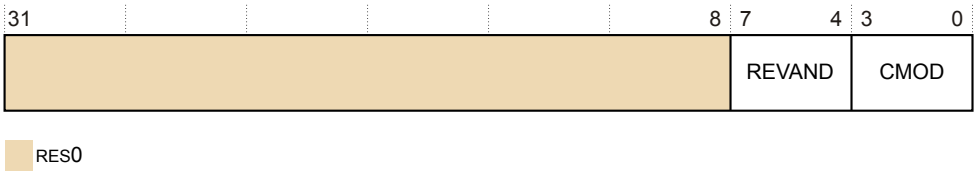

Figure D6-9 PMPIDR3 bit assignments

RES0, [31:8]

RES0 Reserved.

REVAND, [7:4]

`0x0` Part minor revision.

CMOD, [3:0]

`0x0` Customer modified.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The PMPIDR3 can be accessed through the external debug interface, offset `0xFEC`.

D6.11 PMPIDR4, Performance Monitors Peripheral Identification Register 4

The PMPIDR4 provides information to identify a Performance Monitor component.

Bit field descriptions

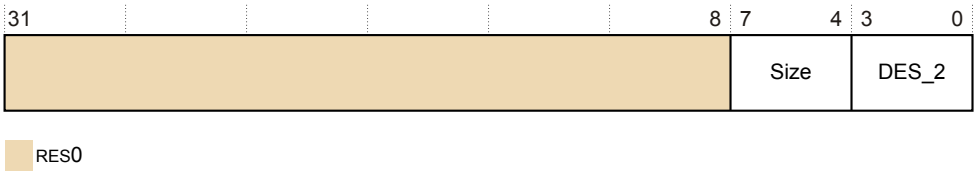

Figure D6-10 PMPIDR4 bit assignments

RES0, [31:8]

RES0 Reserved.

Size, [7:4]

`0x0` Size of the component. Log_2 the number of 4KB pages from the start of the component to the end of the component ID registers.

DES_2, [3:0]

`0x4` Arm Limited. This is the least significant nibble JEP106 continuation code.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The PMPIDR4 can be accessed through the external debug interface, offset `0xFD0`.

D6.12 PMPIDRn, Performance Monitors Peripheral Identification Register 5-7

No information is held in the Peripheral ID5, Peripheral ID6, and Peripheral ID7 Registers.

They are reserved for future use and are RES0.

Chapter D7
AArch64 AMU registers

This chapter describes the AArch64 AMU registers and shows examples of how to use them.

It contains the following sections:

D7.1 AArch64 AMU register summary

The following table gives a summary of the Cortex-A75 AMU registers in the AArch64 Execution state.

Table D7-1 AArch64 AMU registers

Name	Width	Reset	Description
CPUAMCNTENCLR_EL0	32	`0x00000000`	*D7.2 CPUAMCNTENCLR_EL0, Activity Monitors Count Enable Clear Register, EL0* on page D7-661
CPUAMCNTENSET_EL0	32	`0x00000000`	*D7.3 CPUAMCNTENSET_EL0, Activity Monitors Count Enable Set Register, EL0* on page D7-662
CPUAMCFGR_EL0	32	`0x00003F05`	*D7.4 CPUAMCFGR_EL0, Activity Monitors Configuration Register, EL0* on page D7-663
CPUAMUSERENR_EL0	32	`0x00000000`	*D7.5 CPUAMUSERENR_EL0, Activity Monitor EL0 Enable access, EL0* on page D7-665
CPUAMEVCNTR<0-4>_EL0	64	`0x0000000000000000`	*D7.6 CPUAMEVCNTR<0-4>_EL0, Activity Monitor Event Counter Register, EL0* on page D7-667
CPUAMEVTYPER<0-4>_EL0	32	The reset value depends on the register: • CPUAMEVTYPER0_EL0 = `0x00000011`. • CPUAMEVTYPER1_EL0 = `0x000000EF`. • CPUAMEVTYPER2_EL0 = `0x00000008`. • CPUAMEVTYPER<3,4>_EL0 = `0xxxxxxxx`.	*D7.7 CPUAMEVTYPER<0-4>_EL0, Activity Monitor Event Type Register, EL0* on page D7-669

D7.2 CPUAMCNTENCLR_EL0, Activity Monitors Count Enable Clear Register, EL0

The CPUAMCNTENCLR_EL0 disables the activity monitor counters implemented, AMEVCNTR<0-4>.

Usage constraints

This register is accessible as follows:

EL0	EL1	EL2	EL3
RO	RO	RO	RW

Traps and enables

If ACTLR_EL2.CPUAMEN is 0, then Non-secure accesses to this register from EL0 and EL1 are trapped to EL2.

If ACTLR_EL3.CPUAMEN is 0, then accesses to this register from EL0, EL1, and EL2 are trapped to EL3.

If AMUSERENR_EL0.EN is 0, then accesses to this register from EL0 are trapped to EL1.

Configurations

There are no configuration notes.

Attributes

The CPUAMCNTENCLR_EL0 is a 32-bit register.

Field descriptions

P<n>, bit[n]

AMEVCNTR<n> disable bit. The possible values are:

0 When this bit is read, the activity counter n is disabled. When it is written, it has no effect.

1 When this bit is read, the activity counter n is enabled. When it is written, it disables the activity counter n.

Accessing the CPUAMCNTENCLR_EL0

To access the CPUAMCNTENCLR_EL0:

```
MRS <Xt>, CPUAMCNTENCLR_EL0 ; Read CPUAMCNTENCLR_EL0 into Xt
MSR CPUAMCNTENCLR_EL0, <Xt> ; Write <Xt> to CPUAMCNTENCLR_EL0
```

Register access is encoded as follows:

Table D7-2 CPUAMCNTENCLR_EL0 encoding

op0	op1	CRn	CRm	op2
11	011	1111	1001	111

The CPUAMCNTENCLR_EL0 can be accessed through the external debug interface, offset `0xC20`. In this case, it is read-only.

D7.3 CPUAMCNTENSET_EL0, Activity Monitors Count Enable Set Register, EL0

The CPUAMCNTENSET_EL0 enables the activity monitor counters implemented, AMEVCNTR<0-4>.

Usage constraints

This register is accessible as follows:

EL0	EL1	EL2	EL3
RO	RO	RO	RW

Traps and enables

If ACTLR_EL2.CPUAMEN is 0, then Non-secure accesses to this register from EL0 and EL1 are trapped to EL2.

If ACTLR_EL3.CPUAMEN is 0, then accesses to this register from EL0, EL1, and EL2 are trapped to EL3.

If AMUSERENR_EL0.EN is 0, then accesses to this register from EL0 are trapped to EL1.

Configurations

There are no configuration notes.

Attributes

The CPUAMCNTENSET_EL0 is a 32-bit register.

Field descriptions

P<n>, bit[n]

AMEVCNTR<n> enable bit. The possible values are:

0 When this bit is read, the activity counter n is disabled. When it is written, it has no effect.

1 When this bit is read, the activity counter n is enabled. When it is written, it enables the activity counter n.

Accessing the CPUAMCNTENSET_EL0

To access the CPUAMCNTENSET_EL0:

```
MRS <Xt>, CPUAMCNTENSET_EL0 ; Read CPUAMCNTENSET_EL0 into Xt
MSR CPUAMCNTENSET_EL0, <Xt> ; Write <Xt> to CPUAMCNTENSET_EL0
```

Register access is encoded as follows:

Table D7-3 CPUAMCNTENSET_EL0 encoding

op0	op1	CRn	CRm	op2
11	011	1111	1001	110

The CPUAMCNTENSET_EL0 can be accessed through the external debug interface, offset `0xC00`. In this case, it is read-only.

D7.4 CPUAMCFGR_EL0, Activity Monitors Configuration Register, EL0

The CPUAMCFGR_EL0 provides information on the number of activity counters implemented and their size.

Usage constraints

This register is accessible as follows:

EL0	EL1	EL2	EL3
RO	RO	RO	RO

Traps and enables

If ACTLR_EL2.CPUAMEN is 0, then Non-secure accesses to this register from EL0 and EL1 are trapped to EL2.

If ACTLR_EL3.CPUAMEN is 0, then accesses to this register from EL0, EL1, and EL2 are trapped to EL3.

If AMUSERENR_EL0.EN is 0, then accesses to this register from EL0 are trapped to EL1.

Configurations

There are no configuration notes.

Attributes

The CPUAMCFGR_EL0 is a 32-bit register.

The CPUAMCFGR_EL0 bit assignments are:

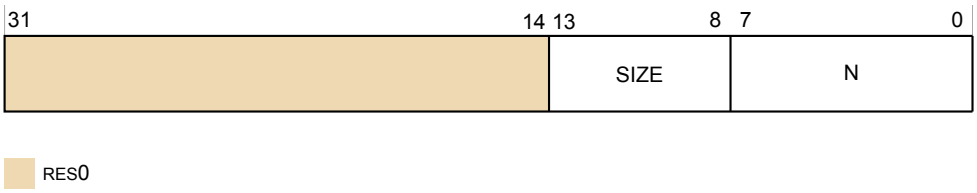

Figure D7-1 CPUAMCFGR_EL0 bit assignments

Field descriptions

RES0, bits[31:14]
Reserved, *RES0*

SIZE, bits[13:8]
Size of counters, minus one.

This field defines the size of the largest counter implemented by the activity monitors. In the Armv8-A architecture, the largest counter has 64 bits, therefore the value of this field is `0b111111`.

N, bits[7:0]
Number of activity counters implemented. The Cortex-A75 core implements five counters, therefore the value is five.

Accessing the CPUAMCFGR_EL0

To access the CPUAMCFGR_EL0:

```
MRS <Xt>, CPUAMCFGR_EL0 ; Read CPUAMCFGR_EL0 into Xt
```

Register access is encoded as follows:

Table D7-4 CPUAMCFGR_EL0 encoding

op0	op1	CRn	CRm	op2
11	011	1111	1010	110

The CPUAMCFGR_EL0 can be accessed through the external debug interface, offset `0xE00`. In this case, it is read-only.

D7.5 CPUAMUSERENR_EL0, Activity Monitor EL0 Enable access, EL0

The CPUAMUSERENR_EL0 enables or disables EL0 access to the activity monitors.

Usage constraints

This register is accessible as follows:

EL0	EL1	EL2	EL3
RO	RW	RW	RW

Note

CPUAMUSERENR_EL0 is always RO at EL0 and not trapped by the EN bit.

Traps and enables

If ACTLR_EL2.CPUAMEN is 0, then Non-secure accesses to this register from EL0 and EL1 are trapped to EL2.

If ACTLR_EL3.CPUAMEN is 0, then accesses to this register from EL0, EL1, and EL2 are trapped to EL3.

Configurations

There are no configuration notes.

Attributes

The CPUAMUSERENR_EL0 is a 32-bit register.

The CPUAMUSERENR_EL0 bit assignments are:

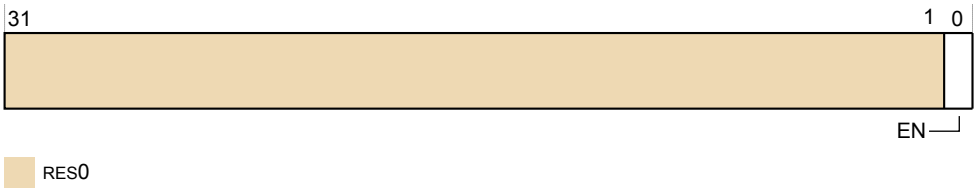

Figure D7-2 CPUAMUSERENR_EL0 bit assignments

Field descriptions

RES0, bits[31:1]
Reserved, RES0.

EN, bit[0]
Traps EL0 accesses to the activity monitor registers to EL1. The possible values are:

0 EL0 accesses to the activity monitor registers are trapped to EL1.

1 EL0 accesses to the activity monitor registers are not trapped to EL1. Software can access all activity monitor registers at EL0.

Accessing the CPUAMUSERENR_EL0

To access the CPUAMUSERENR_EL0:

```
MRS <Xt>, CPUAMUSERENR_EL0 ; Read CPUAMUSERENR_EL0 into Xt
MSR CPUAMUSERENR_EL0, <Xt> ; Write Xt to CPUAMUSERENR_EL0
```

Register access is encoded as follows:

Table D7-5 CPUAMUSERENR_EL0 encoding

op0	op1	CRn	CRm	op2
11	011	1111	1010	111

D7.6 CPUAMEVCNTR<0-4>_EL0, Activity Monitor Event Counter Register, EL0

The activity counters CPUAMEVCNTR<0-4>_EL0 are directly accessible in the memory mapped-view.

Usage constraints

This register is accessible as follows:

EL0	EL1	EL2	EL3
RO	RO	RO	RW

Traps and enables

If ACTLR_EL2.CPUAMEN is 0, then Non-secure accesses to this register from EL0 and EL1 are trapped to EL2.

If ACTLR_EL3.CPUAMEN is 0, then accesses to this register from EL0, EL1, and EL2 are trapped to EL3.

If CPUAMUSERENR_EL0.EN is 0, then accesses to this register from EL0 are trapped to EL1.

Configurations

Counters might have fixed event allocation.

Attributes

The CPUAMEVCNTR<0-4>_EL0 is a 64-bit register.

The CPUAMEVCNTR<0-4>_EL0 bit assignments are:

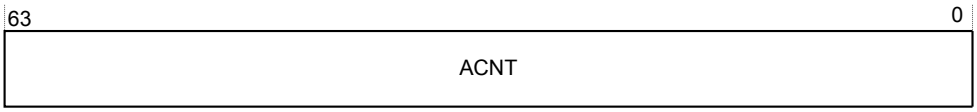

Figure D7-3 CPUAMEVCNTR<0-4>_EL0 bit assignments

Field descriptions

ACNT, bits[63:0]

Value of the activity counter CPUAMEVCNTR<0-4>_EL0.

This bit field resets to zero and the counters monitoring cycle events do not increment when the core is in WFI or WFE.

Accessing the CPUAMEVCNTR<0-4>_EL0

To access the CPUAMEVCNTR<0-4>_EL0:

```
MRS <Xt>, CPUAMEVCNTR<0-4>_EL0 ; Read CPUAMEVCNTR<0-4>_EL0 into Xt
          MSR CPUAMEVCNTR<0-4>_EL0, <Xt> ; Write Xt to CPUAMEVCNTR<0-4>_EL0
```

Register access is encoded as follows:

Table D7-6 CPUAMEVCNTR<0-4>_EL0 encoding

op0	op1	CRn	CRm	op2
11	011	1111	1001	<0-4>

The CPUAMEVCNTR<0-4>_EL0[63:32] can also be accessed through the external memory-mapped interface, offset `0x004+8n`. In this case, it is read-only.

The CPUAMEVCNTR<0-4>_EL0[31:0] can also be accessed through the external memory-mapped interface, offset `0x000+8n`. In this case, it is read-only.

D7.7 CPUAMEVTYPER<0-4>_EL0, Activity Monitor Event Type Register, EL0

The activity counters CPUAMEVTYPER_EL0<0-4> are directly accessible in the memory mapped view.

usage constraints

This register is accessible as follows:

EL0	EL1	EL2	EL3
RO	RO	RO	RW

Traps and enables

If ACTLR_EL2.CPUAMEN is 0, then Non-secure accesses to this register from EL0 and EL1 are trapped to EL2.

If ACTLR_EL3.CPUAMEN is 0, then accesses to this register from EL0, EL1, and EL2 are trapped to EL3.

If CPUAMUSERENR_EL0.EN is 0, then accesses to this register from EL0 are trapped to EL1.

Configurations

Counters might have fixed event allocation.

Attributes

The CPUAMEVTYPER<0-4>_EL0 is a 32-bit register.

The CPUAMEVTYPER<0-4>_EL0 bit assignments are:

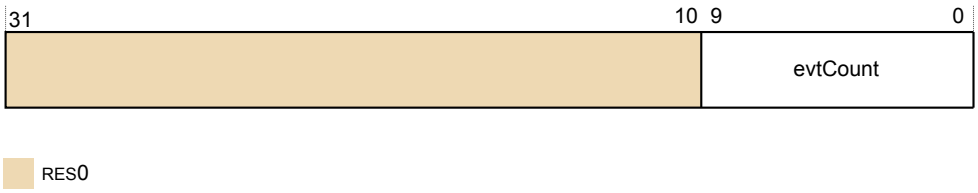

Figure D7-4 CPUAMEVTYPER<0-4>_EL0 bit assignments

Field descriptions

RES0, bits[31:10]
Reserved, RES0.

evtCount, bits[9:0]

Selects the programmable events for counters 3 and 4. The possible values are:

Table D7-7 Programmable events

Event	Event number	Description
L1D_CACHE	0x4	L1 data cache access. This event counts read, write and prefetch accesses to the L1 data cache. This includes non-cacheable speculative reads which do not have cacheability attributes yet. *Cache Maintenance Operation* (CMO) accesses are excluded.
L2D_CACHE	0x16	L2 data cache access. This event counts data reads, instruction reads, and prefetches that hit. Snoops are not counted.
L2D_CACHE_REFILL	0x17	L2 data cache refill. This event counts reads and prefetches that allocate a new linefill buffer entry. The reads and prefetches that fold into a prefetch are not counted.
BUS_ACCESS	0x19	Bus access. This event counts any move of data received from or sent to the SCU.
STALL_FRONTEND	0x23	No operation issued because of the front end. The counter counts every cycle counted by the CPU_CYCLES event on which no operation was issued because there are no operations coming from the instruction side available to issue for this core. Flush windows are excluded.
STALL_BACKEND	0x24	No operation issued because of the back end. The counter counts every cycle counted by the CPU_CYCLES event on which no operation was issued because the rename stage cannot accept any instructions coming from the decoder stage. Flush windows are excluded.

The event the counter monitors might be fixed at implementation. In this case, the field is read-only. See *C3.4 AMU events* on page C3-569.

Accessing the CPUAMEVTYPER<0-4>_EL0

To access the CPUAMEVTYPER<0-4>_EL0:

```
MRS <Xt>, CPUAMEVTYPER<0-4>_EL0 ; Read CPUAMEVTYPER<0-4>_EL0 into Xt
MSR CPUAMEVTYPER<0-4>_EL0, <Xt> ; Write Xt to CPUAMEVTYPER<0-4>_EL0
```

Register access is encoded as follows:

Table D7-8 CPUAMXEVTYPER_EL0 encoding

op0	op1	CRn	CRm	op2
11	011	1111	1010	<0-4>

This register can also be accessed through the external memory-mapped interface, offset 0x400+4n. In this case, it is read-only.

Chapter D8
Memory-mapped AMU registers

This chapter describes the memory-mapped AMU registers and shows examples of how to use them.

It contains the following sections:

D8.1 Memory-mapped AMU register summary

There are AMU registers that are accessible through the external debug interface.

The following table describes the memory-mapped AMU registers implemented in the Cortex-A75 core.

Table D8-1 Memory-mapped AMU register summary

Offset	Name	Type	Description
0x000+8n	CPUAMEVCNTR<0-4>_EL0[31:0]	RO	*D8.2 CPUAMEVCNTR<0-4>_EL0, Activity Monitor Event Counter Register, EL0* on page D8-673 – low bits
0x004+8n	CPUAMEVCNTR<0-4>_EL0[63:32]	RO	*D8.2 CPUAMEVCNTR<0-4>_EL0, Activity Monitor Event Counter Register, EL0* on page D8-673 – high bits
0x400+4n	CPUAMEVTYPER<0-4>_EL0	RO	*D8.3 CPUAMEVTYPER<0-4>_EL0, Activity Monitor Event Type Register, EL0* on page D8-674
-	-	-	Reserved
0xC00	CPUAMCNTENSET_EL0	RO	*D8.4 CPUAMCNTENSET_EL0, Activity Monitor Count Enable Set Register, EL0* on page D8-676
0xC20	CPUAMCNTENCLR_EL0	RO	*D8.5 CPUAMCNTENCLR_EL0, Activity Monitor Count Enable Clear Register, EL0* on page D8-677
-	-	-	Reserved
0xE00	CPUAMCFGR_EL0	RO	*D8.6 CPUAMCFGR, Activity Monitor Configuration Register* on page D8-678

D8.2 CPUAMEVCNTR<0-4>_EL0, Activity Monitor Event Counter Register, EL0

The activity counters CPUAMEVCNTR<0-4>_EL0 are directly accessible in the memory mapped-view.

Usage constraints

This register is read-only.

Attributes

See the register summary in *D8.1 Memory-mapped AMU register summary* on page D8-672.

The CPUAMEVCNTR<0-4>_EL0 bit assignments are:

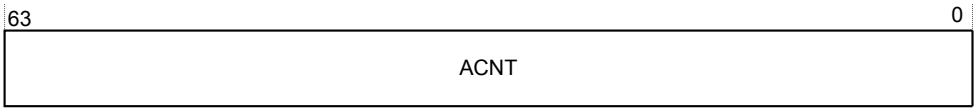

Figure D8-1 CPUAMEVCNTR<0-4>_EL0 bit assignments

Field descriptions

ACNT, bits[63:0]

Value of the activity counter CPUAMEVCNTR<0-4>_EL0.

This bit field resets to zero and the counters monitoring cycle events do not increment when the core is in WFI or WFE.

Accessing the CPUAMEVCNTR<0-4>_EL0

The CPUAMEVCNTR<0-4>_EL0[31:0] can be accessed through the external debug interface, offset `0x000+8n`.

The CPUAMEVCNTR<0-4>_EL0[63:32] can be accessed through the external debug interface, offset `0x004+8n`.

D8.3 CPUAMEVTYPER<0-4>_EL0, Activity Monitor Event Type Register, EL0

The CPUAMEVTYPER_EL0<0-4> are directly accessible in the memory mapped view.

Usage constraints

This register is read-only.

Attributes

See the register summary in *D8.1 Memory-mapped AMU register summary* on page D8-672.

The CPUAMEVTYPER<0-4>_EL0 bit assignments are:

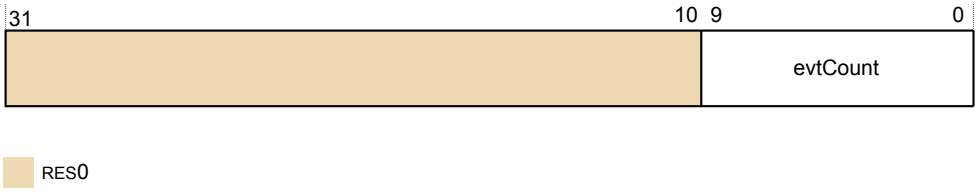

Figure D8-2 CPUAMEVTYPER<0-4>_EL0 bit assignments

Field descriptions

RES0, bits[31:10]
Reserved, RES0.

evtCount, bits[9:0]

Indicates the selected events for the programmable counters. The possible values are:

Table D8-2 Programmable events

Event	Event number	Description
L1D_CACHE	`0x4`	L1 data cache access. This event counts read, write and prefetch accesses to the L1 data cache. This includes non-cacheable speculative reads which do not have cacheability attributes yet. *Cache Maintenance Operation* (CMO) accesses are excluded.
L2D_CACHE	`0x16`	L2 data cache access. This event counts data reads, instruction reads, and prefetches that hit. Snoops are not counted.
L2D_CACHE_REFILL	`0x17`	L2 data cache refill. This event counts reads and prefetches that allocate a new linefill buffer entry. The reads and prefetches that fold into a prefetch are not counted.
BUS_ACCESS	`0x19`	Bus access. This event counts any move of data received from or sent to the SCU.
STALL_FRONTEND	`0x23`	No operation issued because of the front end. The counter counts every cycle counted by the CPU_CYCLES event on which no operation was issued because there are no operations coming from the instruction side available to issue for this core. Flush windows are excluded.
STALL_BACKEND	`0x24`	No operation issued because of the back end. The counter counts every cycle counted by the CPU_CYCLES event on which no operation was issued because the rename stage cannot accept any instructions coming from the decoder stage. Flush windows are excluded.

The event the counter monitors might be fixed at implementation. In this case, the field is read-only. See *C3.4 AMU events* on page C3-569.

Accessing the CPUAMEVTYPER<0-4>_EL0

The CPUAMEVTYPER<0-4>_EL0 can be accessed through the external debug interface, offset `0x400+4n`.

D8.4 CPUAMCNTENSET_EL0, Activity Monitor Count Enable Set Register, EL0

The CPUAMCNTENSET_EL0 enables the activity monitor counters implemented, AMEVCNTR<0-4>.

Usage constraints

This register is read-only.

Attributes

See the register summary in *D8.1 Memory-mapped AMU register summary* on page D8-672.

Field descriptions

P<n>, bit[n]

CPUAMEVCNTR<n> enable bit. The possible values are:

0 When this bit is read, the activity counter n is disabled. When it is written, it has no effect.

1 When this bit is read, the activity counter n is enabled. When it is written, it enables the activity counter n.

Accessing the CPUAMCNTENSET_EL0

The CPUAMCNTENSET_EL0 can be accessed through the external debug interface, offset `0xC00`.

D8.5 CPUAMCNTENCLR_EL0, Activity Monitor Count Enable Clear Register, EL0

The CPUAMCNTENCLR_EL0 disables the activity monitor counters implemented, AMEVCNTR<0-4>.

Usage constraints

This register is read-only.

Attributes

See the register summary in *D8.1 Memory-mapped AMU register summary* on page D8-672.

Field descriptions

P<n>, bit[n]

CPUAMEVCNTR<n> disable bit. The possible values are:

0 When this bit is read, the activity counter n is disabled. When it is written, it has no effect.

1 When this bit is read, the activity counter n is enabled. When it is written, it disables the activity counter n.

Accessing the CPUAMCNTENCLR_EL0

The CPUAMCNTENCLR_EL0 can be accessed through the external debug interface, offset `0xC20`.

D8.6 CPUAMCFGR, Activity Monitor Configuration Register

The CPUAMCFGR provides information on the number of activity counters implemented and their size.

Usage constraints

This register is read-only.

Attributes

See the register summary in *D8.1 Memory-mapped AMU register summary* on page D8-672.

The CPUAMCFGR bit assignments are:

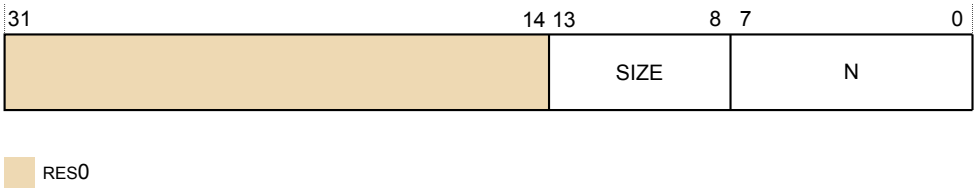

Figure D8-3 CPUAMCFGR bit assignments

Field descriptions

RES0, bits[31:14]
Reserved, *RES0*

SIZE, bits[13:8]
Size of counters, minus one.

This field defines the size of the largest counter implemented by the activity monitors. In the Armv8-A architecture, the largest counter has 64 bits, therefore the value of this field is `0b111111`.

N, bits[7:0]
Number of activity counters implemented, where the number of counters is N+1. The Cortex-A75 core implements five counters, therefore the value is five.

Accessing the CPUAMCFGR

The CPUAMCFGR can be accessed through the external debug interface, offset `0xE00`.

Chapter D9
PMU snapshot registers

PMU snapshot registers are an implementation defined extension to an Armv8 compliant PMU to support an external core monitor that connects to a system profiler.

It contains the following sections:

D9.1 PMU snapshot register summary

The snapshot registers are visible in an implementation defined region of the PMU external debug interface. Each time the debugger sends a snapshot request, information is collected to see how the code is executed in the different cores.

The following table describes the PMU snapshot registers implemented in the core.

Table D9-1 PMU snapshot register summary

Offset	Name	Type	Width	Description
0x600	PMPCSSR_LO	RO	32	*D9.2 PMPCSSR, Snapshot Program Counter Sample Register* on page D9-681
0x604	PMPCSSR_HI	RO	32	
0x608	PMPCIDSSR	RO	32	*D9.3 PMCIDSSR, Snapshot CONTEXTIDR_EL1 Sample Register* on page D9-682
0x60C	PMPCID2SSR	RO	32	*D9.4 PMCID2SSR, Snapshot CONTEXTIDR_EL2 Sample Register* on page D9-683
0x610	PMSSSR	RO	32	*D9.5 PMSSSR, PMU Snapshot Status Register* on page D9-684
0x614	PMOVSSR	RO	32	*D9.6 PMOVSSR, PMU Overflow Status Snapshot Register* on page D9-685
0x618	PMCCNTSR_LO	RO	32	*D9.7 PMCCNTSR, PMU Cycle Counter Snapshot Register* on page D9-686
0x61C	PMCCNTSR_HI	RO	32	
0x620 + 4×n	PMEVCNTSR<n>	RO	32	*D9.8 PMEVCNTSR 0-5, PMU Cycle Counter Snapshot Registers* on page D9-687
0x6F0	PMSSCR	WO	32	*D9.9 PMSSCR, PMU Snapshot Capture Register* on page D9-688

D9.2 PMPCSSR, Snapshot Program Counter Sample Register

The PMPCSSR is an alias for the PCSR register.

However, unlike the other view of PCSR, it is not sensitive to reads. That is, reads of PMPCSSR through the PMU snapshot view do not cause a new sample capture and do not change CIDSR, CID2SR, or VIDSR.

Bit field descriptions

The PMPCSSR is a 64-bit read-only register.

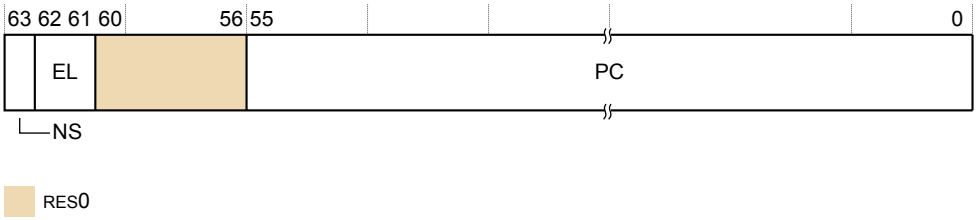

Figure D9-1 PMPCSSR bit assignments

NS, [63]
Non-secure sample.

EL, [62:61]
Exception level sample.

[60:56]
Reserved, RES0.

PC, [55:0]
Sampled PC.

Configurations
There are no configuration notes.

Usage constraints

Any access to PMPCSSR returns an error if any of the following occurs:

- The core power domain is off.
- DoubleLockStatus() == TRUE.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

D9.3 PMCIDSSR, Snapshot CONTEXTIDR_EL1 Sample Register

The PMCIDSSR is an alias for the CIDSR register.

Configurations

There are no configuration notes.

Usage constraints

Any access to PMCIDSSR returns an error if any of the following occurs:

- The core power domain is off.
- DoubleLockStatus() == TRUE.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

D9.4 PMCID2SSR, Snapshot CONTEXTIDR_EL2 Sample Register

The PMCID2SSR is an alias for the CID2SR register.

Configurations

There are no configuration notes.

Usage constraints

Any access to PMCID2SSR returns an error if any of the following occurs:

- The core power domain is off.
- DoubleLockStatus() == TRUE.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

D9.5 PMSSSR, PMU Snapshot Status Register

The PMSSSR holds status information about the captured counters.

Bit field descriptions

The PMSSSR is a 32-bit read-only register.

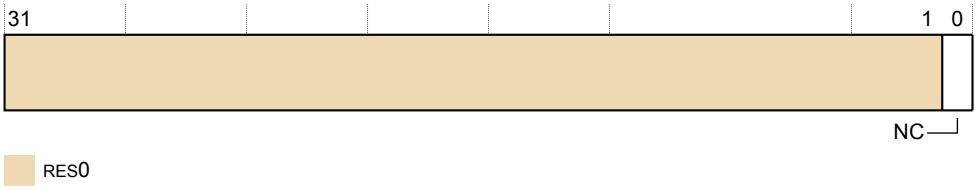

Figure D9-2 PMSSSR bit assignments

[31:1]

Reserved, RES0.

NC, [0]

No capture. This bit indicates whether the PMU counters have been captured. The possible values are:

0 PMU counters captured.

1 PMU counters not captured.

The core does not capture the event counters only if there is a security violation. The external monitor is responsible for keeping track of whether it managed to capture the snapshot registers from the core.

This bit does not reflect the status of the captured Program Counter Sample registers.

The core resets this bit to 1 by a Warm reset but MPSSSR.NC is overwritten at the first capture.

Configurations

There are no configuration notes.

Usage constraints

Any access to PMSSSR returns an error if any of the following occurs:

- The core power domain is off.
- DoubleLockStatus() == TRUE.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

D9.6 PMOVSSR, PMU Overflow Status Snapshot Register

The PMOVSSR is a captured copy of PMOVSR.

Once it is captured, the value in PMOVSSR is unaffected by writes to PMOVSSET_EL0 and PMOVSCLR_EL0.

Configurations

There are no configuration notes.

Usage constraints

Any access to PMOVSSR returns an error if any of the following occurs:

- The core power domain is off.
- DoubleLockStatus() == TRUE.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

D9.7 PMCCNTSR, PMU Cycle Counter Snapshot Register

The PMCCNTSR is a captured copy of PMCCNTR_EL0.

Once it is captured, the value in PMCCNTSR is unaffected by writes to PMCCNTR_EL0 and PMCR_EL0.C.

Configurations

There are no configuration notes.

Usage constraints

Any access to PMCCNTSR returns an error if any of the following occurs:

- The core power domain is off.
- DoubleLockStatus() == TRUE.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

D9.8 PMEVCNTSR 0-5, PMU Cycle Counter Snapshot Registers

The PMEVCNTSR 0-5 are captured copies of PMEVCNTR<n>_EL0.

Once they are captured, the value in PMSSEVCNTR<n> is unaffected by writes to PMSSEVCNTR<n>_EL0 and PMCR_EL0.P.

Configurations

There are no configuration notes.

Usage constraints

Any access to PMSSEVCNTR 0-5 returns an error if any of the following occurs:

- The core power domain is off.
- DoubleLockStatus() == TRUE.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

D9.9 PMSSCR, PMU Snapshot Capture Register

The PMSSCR provides a mechanism for software to initiate a sample.

Bit field descriptions

The PMSSCR is a 32-bit write-only register.

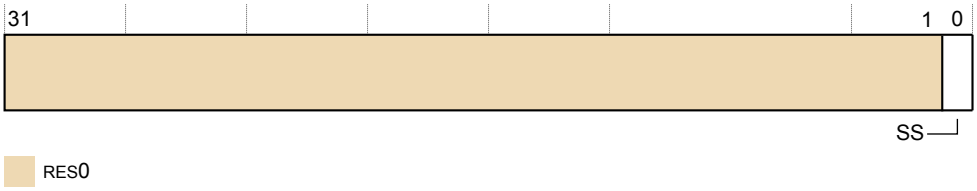

Figure D9-3 PMSSCR bit assignments

[31:1]

Reserved, RES0.

SS, [0]

Capture now. The possible values are:

0 Ignored.

1 Initiate a capture immediately.

Configurations

There are no configuration notes.

Usage constraints

Any access to PMSSCR returns an error if any of the following occurs:

- The core power domain is off.
- DoubleLockStatus() == TRUE.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

Chapter D10
ETM registers

This chapter describes the ETM registers.

It contains the following sections:

D10.1 ETM register summary

This section summarizes the ETM trace unit registers.

All ETM trace unit registers are 32-bit wide. The description of each register includes its offset from a base address. The base address is defined by the system integrator when placing the ETM trace unit in the Debug-APB memory map.

The following table lists all of the ETM trace unit registers.

Table D10-1 ETM trace unit register summary

Offset	Name	Type	Reset	Description
0x004	TRCPRGCTLR	RW	0x00000000	*D10.60 TRCPRGCTLR, Programming Control Register* on page D10-764
0x00C	TRCSTATR	RO	0x00000003	*D10.68 TRCSTATR, Status Register* on page D10-773
0x010	TRCCONFIGR	RW	UNK	*D10.20 TRCCONFIGR, Trace Configuration Register* on page D10-717
0x018	TRCAUXCTLR	RW	0x00000000	*D10.5 TRCAUXCTLR, Auxiliary Control Register* on page D10-699
0x020	TRCEVENTCTL0R	RW	UNK	*D10.26 TRCEVENTCTL0R, Event Control 0 Register* on page D10-725
0x024	TRCEVENTCTL1R	RW	UNK	*D10.27 TRCEVENTCL1R, Event Control 1 Register* on page D10-727
0x02C	TRCSTALLCTLR	RW	UNK	*D10.67 TRCSTALLCTLR, Stall Control Register* on page D10-772
0x030	TRCTSCTLR	RW	UNK	*D10.71 TRCTSCTLR, Global Timestamp Control Register* on page D10-776
0x034	TRCSYNCPR	RW	UNK	*D10.69 TRCSYNCPR, Synchronization Period Register* on page D10-774
0x038	TRCCCCTLR	RW	UNK	*D10.7 TRCCCCTLR, Cycle Count Control Register* on page D10-702
0x03C	TRCBBCTLR	RW	UNK	*D10.6 TRCBBCTLR, Branch Broadcast Control Register* on page D10-701
0x040	TRCTRACEIDR	RW	UNK	*D10.70 TRCTRACEIDR, Trace ID Register* on page D10-775
0x080	TRCVICTLR	RW	UNK	*D10.72 TRCVICTLR, ViewInst Main Control Register* on page D10-777
0x084	TRCVIIECTLR	RW	UNK	*D10.73 TRCVIIECTLR, ViewInst Include-Exclude Control Register* on page D10-779
0x088	TRCVISSCTLR	RW	UNK	*D10.74 TRCVISSCTLR, ViewInst Start-Stop Control Register* on page D10-780
0x100	TRCSEQEVR0	RW	UNK	*D10.62 TRCSEQEVRn, Sequencer State Transition Control Registers 0-2* on page D10-766
0x104	TRCSEQEVR1	RW	UNK	*D10.62 TRCSEQEVRn, Sequencer State Transition Control Registers 0-2* on page D10-766
0x108	TRCSEQEVR2	RW	UNK	*D10.62 TRCSEQEVRn, Sequencer State Transition Control Registers 0-2* on page D10-766
0x118	TRCSEQRSTEVR	RW	UNK	*D10.63 TRCSEQRSTEVR, Sequencer Reset Control Register* on page D10-768
0x11C	TRCSEQSTR	RW	UNK	*D10.64 TRCSEQSTR, Sequencer State Register* on page D10-769
0x120	TRCEXTINSELR	RW	UNK	*D10.28 TRCEXTINSELR, External Input Select Register* on page D10-728
0x140	TRCCNTRLDVR0	RW	UNK	*D10.18 TRCCNTRLDVRn, Counter Reload Value Registers 0-1* on page D10-715

Table D10-1 ETM trace unit register summary (continued)

Offset	Name	Type	Reset	Description
0x144	TRCCNTRLDVR1	RW	UNK	*D10.18 TRCCNTRLDVRn, Counter Reload Value Registers 0-1* on page D10-715
0x150	TRCCNTCTLR0	RW	UNK	*D10.16 TRCCNTCTLR0, Counter Control Register 0* on page D10-711
0x154	TRCCNTCTLR1	RW	UNK	*D10.17 TRCCNTCTLR1, Counter Control Register 1* on page D10-713
0x160	TRCCNTVR0	RW	UNK	*D10.19 TRCCNTVRn, Counter Value Registers 0-1* on page D10-716
0x164	TRCCNTVR1	RW	UNK	*D10.19 TRCCNTVRn, Counter Value Registers 0-1* on page D10-716
0x180	TRCIDR8	RO	0x00000000	*D10.35 TRCIDR8, ID Register 8* on page D10-739
0x184	TRCIDR9	RO	0x00000000	*D10.36 TRCIDR9, ID Register 9* on page D10-740
0x188	TRCIDR10	RO	0x00000000	*D10.37 TRCIDR10, ID Register 10* on page D10-741
0x18C	TRCIDR11	RO	0x00000000	*D10.38 TRCIDR11, ID Register 11* on page D10-742
0x190	TRCIDR12	RO	0x00000000	*D10.39 TRCIDR12, ID Register 12* on page D10-743
0x194	TRCIDR13	RO	0x00000000	*D10.40 TRCIDR13, ID Register 13* on page D10-744
0x1C0	TRCIMSPEC0	RW	0x00000000	*D10.41 TRCIMSPEC0, Implementation Specific Register 0* on page D10-745
0x1E0	TRCIDR0	RO	0x28000EA1	*D10.29 TRCIDR0, ID Register 0* on page D10-729
0x1E4	TRCIDR1	RO	0x4100F422	*D10.30 TRCIDR1, ID Register 1* on page D10-731
0x1E8	TRCIDR2	RO	0x20001088	*D10.31 TRCIDR2, ID Register 2* on page D10-732
0x1EC	TRCIDR3	RO	0x0D7B0004	*D10.32 TRCIDR3, ID Register 3* on page D10-734
0x1F0	TRCIDR4	RO	0x11170004	*D10.33 TRCIDR4, ID Register 4* on page D10-736
0x1F4	TRCIDR5	RO	0x28C70820	*D10.34 TRCIDR5, ID Register 5* on page D10-737
0x200	TRCRSCTLRn	RW	UNK	*D10.61 TRCRSCTLRn, Resource Selection Control Registers 2-16* on page D10-765, n is 2, 15
0x280	TRCSSCCR0	RW	UNK	*D10.65 TRCSSCCR0, Single-Shot Comparator Control Register 0* on page D10-770
0x2A0	TRCSSCSR0	RW	UNK	*D10.66 TRCSSCSR0, Single-Shot Comparator Status Register 0* on page D10-771
0x300	TRCOSLAR	WO	0x00000001	*D10.50 TRCOSLAR, OS Lock Access Register* on page D10-754
0x304	TRCOSLSR	RO	0x0000000A	*D10.51 TRCOSLSR, OS Lock Status Register* on page D10-755
0x310	TRCPDCR	RW	0x00000000	*D10.52 TRCPDCR, Power Down Control Register* on page D10-756
0x314	TRCPDSR	RO	0x00000013	*D10.53 TRCPDSR, Power Down Status Register* on page D10-757
0x400	TRCACVRn	RW	UNK	*D10.3 TRCACVRn, Address Comparator Value Registers 0-7* on page D10-697
0x480	TRCACATRn	RW	UNK	*D10.2 TRCACATRn, Address Comparator Access Type Registers 0-7* on page D10-695
0x600	TRCCIDCVR0	RW	UNK	*D10.9 TRCCIDCVR0, Context ID Comparator Value Register 0* on page D10-704

Table D10-1 ETM trace unit register summary (continued)

Offset	Name	Type	Reset	Description
0x640	TRCVMIDCVR0	RW	UNK	*D10.75 TRCVMIDCVR0, VMID Comparator Value Register 0* on page D10-781
0x680	TRCCIDCCTLR0	RW	UNK	*D10.8 TRCCIDCCTLR0, Context ID Comparator Control Register 0* on page D10-703
0x688	TRCVMIDCCTRL0	RW	UNK	Virtual context identifier Comparator Control Register 0
0xEE4	TRCITATBIDR	RW	UNK	*D10.42 TRCITATBIDR, Integration ATB Identification Register* on page D10-746
0xEEC	TRCITIDATAR	WO	UNK	*D10.46 TRCITIDATAR, Integration Instruction ATB Data Register* on page D10-750
0xEF4	TRCITIATBINR	RO	UNK	*D10.44 TRCITIATBINR, Integration Instruction ATB In Register* on page D10-748
0xEFC	TRCITIATBOUTR	WO	UNK	*D10.45 TRCITIATBOUTR, Integration Instruction ATB Out Register* on page D10-749
0xF00	TRCITCTRL	RW	0x00000000	*D10.43 TRCITCTRL, Integration Mode Control Register* on page D10-747
0xFA0	TRCCLAIMSET	RW	UNK	*D10.15 TRCCLAIMSET, Claim Tag Set Register* on page D10-710
0xFA4	TRCCLAIMCLR	RW	0x00000000	*D10.14 TRCCLAIMCLR, Claim Tag Clear Register* on page D10-709
0xFA8	TRCDEVAFF0	RO	UNK	*D10.21 TRCDEVAFF0, Device Affinity Register 0* on page D10-719
0xFAC	TRCDEVAFF1	RO	UNK	*D10.22 TRCDEVAFF1, Device Affinity Register 1* on page D10-721
0xFB0	TRCLAR	WO	UNK	*D10.47 TRCLAR, Software Lock Access Register* on page D10-751
0xFB4	TRCLSR	RO	0x00000000	*D10.48 TRCLSR, Software Lock Status Register* on page D10-752
0xFB8	TRCAUTHSTATUS	RO	UNK	*D10.4 TRCAUTHSTATUS, Authentication Status Register* on page D10-698
0xFBC	TRCDEVARCH	RO	0x47724A13	*D10.23 TRCDEVARCH, Device Architecture Register* on page D10-722
0xFC8	TRCDEVID	RO	0x00000000	*D10.24 TRCDEVID, Device ID Register* on page D10-723
0xFCC	TRCDEVTYPE	RO	0x00000013	*D10.25 TRCDEVTYPE, Device Type Register* on page D10-724
0xFE0	TRCPIDR0	RO	0x0000000A	*D10.54 TRCPIDR0, ETM Peripheral Identification Register 0* on page D10-758
0xFE4	TRCPIDR1	RO	0x000000BD	*D10.55 TRCPIDR1, ETM Peripheral Identification Register 1* on page D10-759
0xFE8	TRCPIDR2	RO	0x0200000B	*D10.56 TRCPIDR2, ETM Peripheral Identification Register 2* on page D10-760
0xFEC	TRCPIDR3	RO	0x00000000	*D10.57 TRCPIDR3, ETM Peripheral Identification Register 3* on page D10-761
0xFD0	TRCPIDR4	RO	0x00000004	*D10.58 TRCPIDR4, ETM Peripheral Identification Register 4* on page D10-762
0xFD4-0xFDC	TRCPIDRn	RO	0x00000000	*D10.59 TRCPIDRn, ETM Peripheral Identification Registers 5-7* on page D10-763
0xFF0	TRCCIDR0	RO	0x0000000D	*D10.10 TRCCIDR0, ETM Component Identification Register 0* on page D10-705

Table D10-1 ETM trace unit register summary (continued)

Offset	Name	Type	Reset	Description
0xFF4	TRCCIDR1	RO	0x00000090	*D10.11 TRCCIDR1, ETM Component Identification Register 1* on page D10-706
0xFF8	TRCCIDR2	RO	0x00000005	*D10.12 TRCCIDR2, ETM Component Identification Register 2* on page D10-707
0xFFC	TRCCIDR3	RO	0x000000B1	*D10.13 TRCCIDR3, ETM Component Identification Register 3* on page D10-708

D10.2 TRCACATRn, Address Comparator Access Type Registers 0-7

The TRCACATRn control the access for the corresponding address comparators.

Bit field descriptions

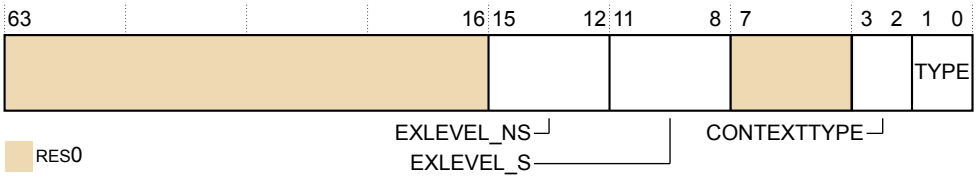

Figure D10-1 TRCACATRn bit assignments

RES0, [63:16]

RES0 Reserved.

EXLEVEL_NS, [15:12]

Each bit controls whether a comparison can occur in Non-secure state for the corresponding exception level. The possible values are:

`0` The trace unit can perform a comparison, in Non-secure state, for exception level *n*.

`1` The trace unit does not perform a comparison, in Non-secure state, for exception level *n*.

The Exception levels are:

Bit[12] Exception level 0.

Bit[13] Exception level 1.

Bit[14] Exception level 2.

Bit[15] Always RES0.

EXLEVEL_S, [11:8]

Each bit controls whether a comparison can occur in Secure state for the corresponding exception level. The possible values are:

`0` The trace unit can perform a comparison, in Secure state, for exception level *n*.

`1` The trace unit does not perform a comparison, in Secure state, for exception level *n*.

The Exception levels are:

Bit[8] Exception level 0.

Bit[9] Exception level 1.

Bit[10] Always RES0.

Bit[11] Exception level 3.

RES0, [7:4]

RES0 Reserved.

CONTEXT TYPE, [3:2]

Controls whether the trace unit performs a Context ID comparison, a VMID comparison, or both comparisons:

`0b00` The trace unit does not perform a Context ID comparison.

`0b01` The trace unit performs a Context ID comparison using the Context ID comparator that the CONTEXT field specifies, and signals a match if both the Context ID comparator matches and the address comparator match.

`0b10` The trace unit performs a VMID comparison using the VMID comparator that the CONTEXT field specifies, and signals a match if both the VMID comparator and the address comparator match.

`0b11` The trace unit performs a Context ID comparison and a VMID comparison using the comparators that the CONTEXT field specifies, and signals a match if the Context ID comparator matches, the VMID comparator matches, and the address comparator matches.

TYPE, [1:0]

Type of comparison:

`0b00` Instruction address, RES0.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The TRCACATR*n* can be accessed through the external debug interface, offset `0x480-0x4B8`.

D10.3 TRCACVRn, Address Comparator Value Registers 0-7

The TRCACVRn indicate the address for the address comparators.

Bit field descriptions

Figure D10-2 TRCACVRn bit assignments

ADDRESS, [63:0]
The address value to compare against.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The TRCACVRn can be accessed through the external debug interface, offset `0x400-0x43C`.

D10.4 TRCAUTHSTATUS, Authentication Status Register

The TRCAUTHSTATUS indicates the current level of tracing permitted by the system.

Bit field descriptions

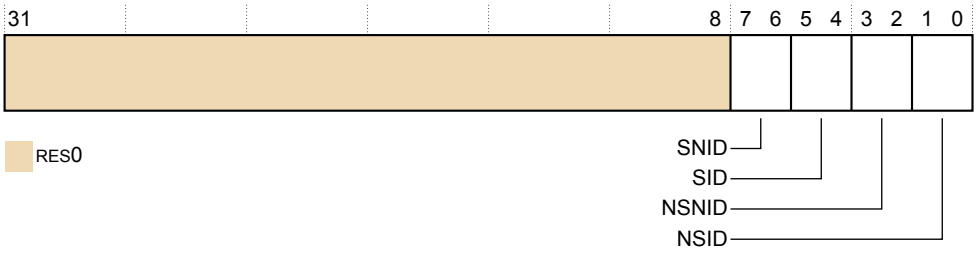

Figure D10-3 TRCAUTHSTATUS bit assignments

RES0, [31:8]

RES0 Reserved.

SNID, [7:6]

Secure Non-invasive Debug:

`0b10` Secure Non-invasive Debug implemented but disabled.

`0b11` Secure Non-invasive Debug implemented and enabled.

SID, [5:4]

Secure Invasive Debug:

`0b00` Secure Invasive Debug is not implemented.

NSNID, [3:2]

Non-secure Non-invasive Debug:

`0b10` Non-secure Non-invasive Debug implemented but disabled, **NIDEN**=0.

`0b11` Non-secure Non-invasive Debug implemented and enabled, **NIDEN**=1.

NSID, [1:0]

Non-secure Invasive Debug:

`0b00` Non-secure Invasive Debug is not implemented.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The TRCAUTHSTATUS can be accessed through the external debug interface, offset `0xFB8`.

D10.5 TRCAUXCTLR, Auxiliary Control Register

The TRCAUXCTLR provides IMPLEMENTATION DEFINED configuration and control options.

Bit field descriptions

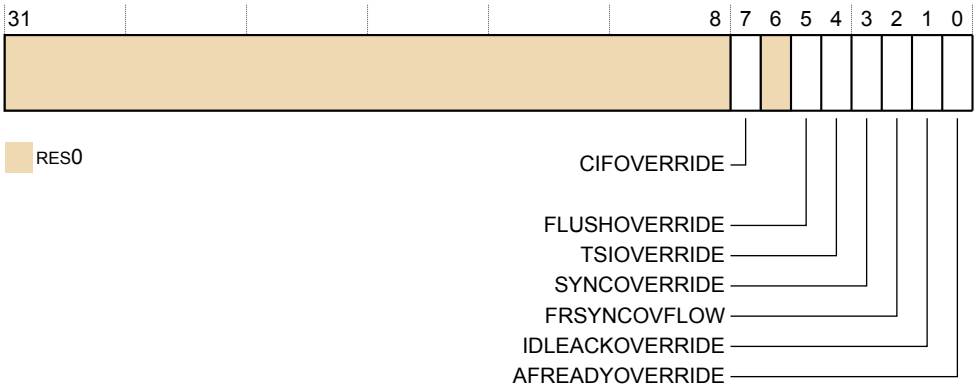

Figure D10-4 TRCAUXCTLR bit assignments

RES0, [31:8]

RES0 Reserved.

CIFOVERRIDE, [7]

Override core interface register repeater clock enable. The possible values are:

0 Core interface clock gate is enabled.

1 Core interface clock gate is disabled.

RES0, [6]

RES0 Reserved.

FLUSHOVERRIDE, [5]

Override ETM flush behavior. The possible values are:

0 ETM trace unit FIFO is flushed and ETM trace unit enters idle state when **DBGEN** or **NIDEN** is LOW.

1 ETM trace unit FIFO is not flushed and ETM trace unit does not enter idle state when **DBGEN** or **NIDEN** is LOW.

When this bit is set to 1, the trace unit behavior deviates from architecturally-specified behavior.

TSIOVERRIDE, [4]

Override TS packet insertion behavior. The possible values are:

0 Timestamp packets are inserted into FIFO only when trace activity is LOW.

1 Timestamp packets are inserted into FIFO irrespective of trace activity.

SYNCOVERRIDE, [3]

Override SYNC packet insertion behavior. The possible values are:

0 SYNC packets are inserted into FIFO only when trace activity is low.

1 SYNC packets are inserted into FIFO irrespective of trace activity.

FRSYNCOVFLOW, [2]

Force overflows to output synchronization packets. The possible values are:

0 No FIFO overflow when SYNC packets are delayed.

`1` Forces FIFO overflow when SYNC packets are delayed.

When this bit is set to 1, the trace unit behavior deviates from architecturally-specified behavior.

IDLEACKOVERRIDE, [1]
Force ETM idle acknowledge. The possible values are:

`0` ETM trace unit idle acknowledge is asserted only when the ETM trace unit is in idle state.

`1` ETM trace unit idle acknowledge is asserted irrespective of the ETM trace unit idle state.

When this bit is set to 1, trace unit behavior deviates from architecturally-specified behavior.

AFREADYOVERRIDE, [0]
Force assertion of **AFREADYM** output. The possible values are:

`0` ETM trace unit **AFREADYM** output is asserted only when the ETM trace unit is in idle state or when all the trace bytes in FIFO before a flush request are output.

`1` ETM trace unit **AFREADYM** output is always asserted HIGH.

When this bit is set to 1, trace unit behavior deviates from architecturally-specified behavior.

The TRCAUXCTLR can be accessed through the internal memory-mapped interface and the external debug interface, offset `0x018`.

Configurations
Available in all configurations.

Attributes
See *D10.1 ETM register summary* on page D10-691.

D10.6 TRCBBCTLR, Branch Broadcast Control Register

The TRCBBCTLR controls how branch broadcasting behaves, and enables branch broadcasting to be enabled for certain memory regions.

Bit field descriptions

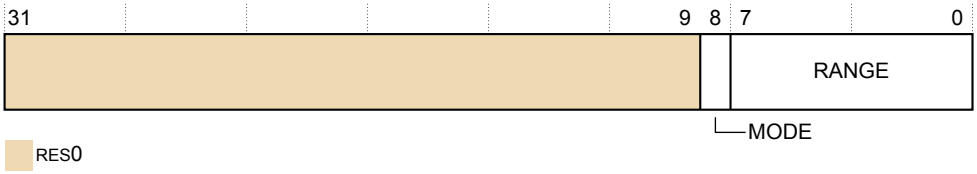

Figure D10-5 TRCBBCTLR bit assignments

RES0, [31:9]

RES0 Reserved.

MODE, [8]

Mode bit:

`0` Exclude mode. Branch broadcasting is not enabled in the address range that RANGE defines.

If RANGE==0 then branch broadcasting is enabled for the entire memory map.

`1` Include mode. Branch broadcasting is enabled in the address range that RANGE defines.

If RANGE==0 then the behavior of the trace unit is constrained UNPREDICTABLE. That is, the trace unit might or might not consider any instructions to be in a branch broadcast region.

RANGE, [7:0]

Address range field.

Selects which address range comparator pairs are in use with branch broadcasting. Each bit represents an address range comparator pair, so bit[*n*] controls the selection of address range comparator pair *n*. If bit[*n*] is:

`0` The address range that address range comparator pair *n* defines, is not selected.

`1` The address range that address range comparator pair *n* defines, is selected.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The TRCBBCTLR can be accessed through the external debug interface, offset `0x03C`.

D10.7 TRCCCCTLR, Cycle Count Control Register

The TRCCCCTLR sets the threshold value for cycle counting.

Bit field descriptions

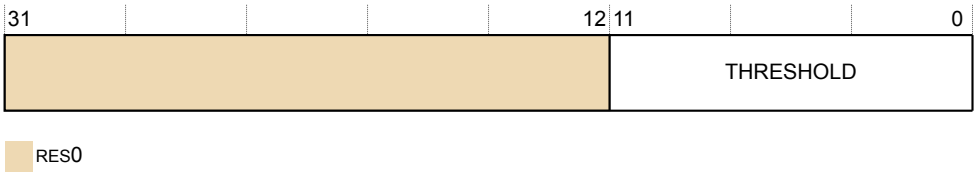

Figure D10-6 TRCCCCTLR bit assignments

RES0, [31:12]

RES0 Reserved.

THRESHOLD, [11:0]

Instruction trace cycle count threshold.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The TRCCCCTLR can be accessed through the external debug interface, offset `0x038`.

D10.8 TRCCIDCCTLR0, Context ID Comparator Control Register 0

The TRCCIDCCTLR0 controls the mask value for the context ID comparators.

Bit field descriptions

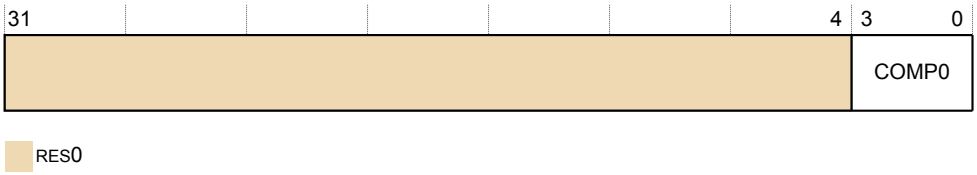

Figure D10-7 TRCCIDCCTLR0 bit assignments

RES0, [31:4]

RES0 Reserved.

COMP0, [3:0]

Controls the mask value that the trace unit applies to TRCCIDCVR0. Each bit in this field corresponds to a byte in TRCCIDCVR0. When a bit is:

`0` The trace unit includes the relevant byte in TRCCIDCVR0 when it performs the Context ID comparison.

`1` The trace unit ignores the relevant byte in TRCCIDCVR0 when it performs the Context ID comparison.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The TRCCIDCCTLR0 can be accessed through the external debug interface, offset `0x680`.

D10.9 TRCCIDCVR0, Context ID Comparator Value Register 0

The TRCCIDCVR0 contains a Context ID value.

Bit field descriptions

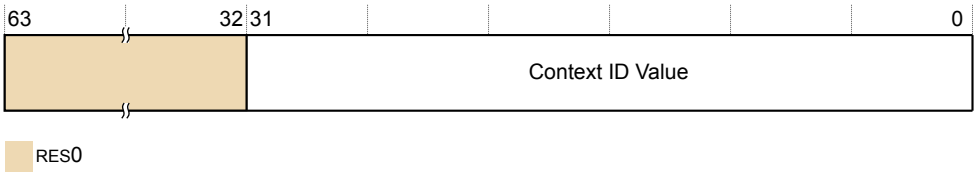

Figure D10-8 TRCCIDCVR0 bit assignments

RES0, [63:32]

RES0 Reserved.

VALUE, [31:0]

The data value to compare against.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The TRCCIDCVR0 can be accessed through the external debug interface, offset `0x600`.

D10.10 TRCCIDR0, ETM Component Identification Register 0

The TRCCIDR0 provides information to identify a trace component.

Bit field descriptions

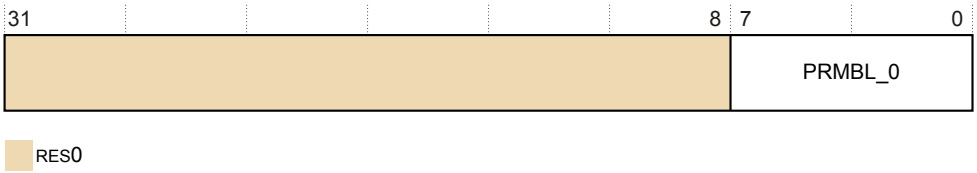

Figure D10-9 TRCCIDR0 bit assignments

RES0, [31:8]

RES0 Reserved.

PRMBL_0, [7:0]

`0x0D` Preamble byte 0.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The TRCCIDR0 can be accessed through the external debug interface, offset `0xFF0`.

D10.11 TRCCIDR1, ETM Component Identification Register 1

The TRCCIDR1 provides information to identify a trace component.

Bit field descriptions

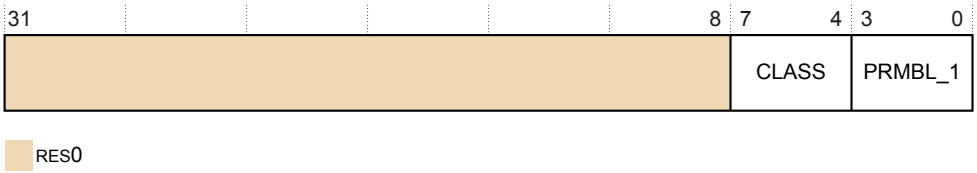

Figure D10-10 TRCCIDR1 bit assignments

RES0, [31:8]

RES0 Reserved.

CLASS, [7:4]

`0x9` Debug component.

PRMBL_1, [3:0]

`0x0` Preamble byte 1.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The TRCCIDR1 can be accessed through the external debug interface, offset `0xFF4`.

D10.12 TRCCIDR2, ETM Component Identification Register 2

The TRCCIDR2 provides information to identify a CTI component.

Bit field descriptions

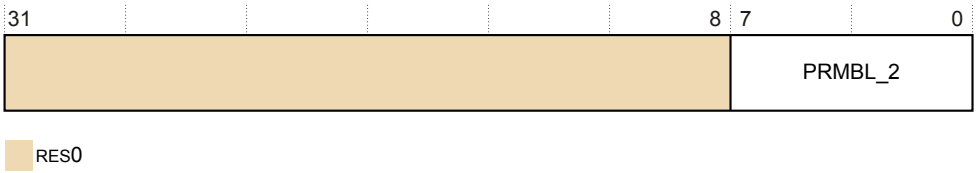

Figure D10-11 TRCCIDR2 bit assignments

RES0, [31:8]

RES0 Reserved.

PRMBL_2, [7:0]

`0x05` Preamble byte 2.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The TRCCIDR2 can be accessed through the external debug interface, offset `0xFF8`.

D10.13 TRCCIDR3, ETM Component Identification Register 3

The TRCCIDR3 provides information to identify a trace component.

Bit field descriptions

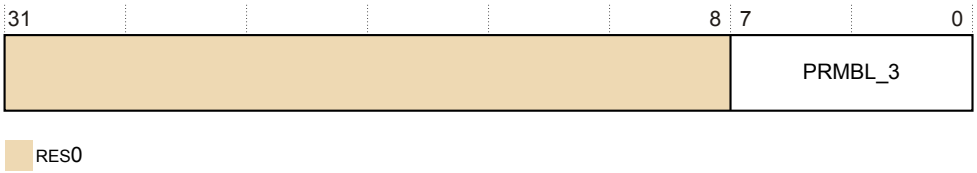

Figure D10-12 TRCCIDR3 bit assignments

RES0, [31:8]

RES0 Reserved.

PRMBL_3, [7:0]

`0xB1` Preamble byte 3.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The TRCCIDR3 can be accessed through the external debug interface, offset `0xFFC`.

D10.14 TRCCLAIMCLR, Claim Tag Clear Register

The TRCCLAIMCLR clears bits in the claim tag and determines the current value of the claim tag.

Bit field descriptions

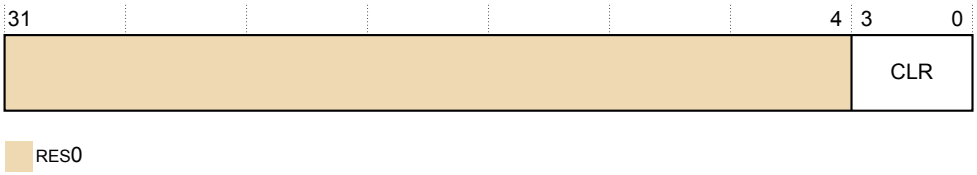

Figure D10-13 TRCCLAIMCLR bit assignments

RES0, [31:4]

RES0 Reserved.

CLR, [3:0]

On reads, for each bit:

`0` Claim tag bit is not set.

`1` Claim tag bit is set.

On writes, for each bit:

`0` Has no effect.

`1` Clears the relevant bit of the claim tag.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The TRCCLAIMCLR can be accessed through the external debug interface, offset `0xFA4`.

D10.15 TRCCLAIMSET, Claim Tag Set Register

The TRCCLAIMSET sets bits in the claim tag and determines the number of claim tag bits implemented.

Bit field descriptions

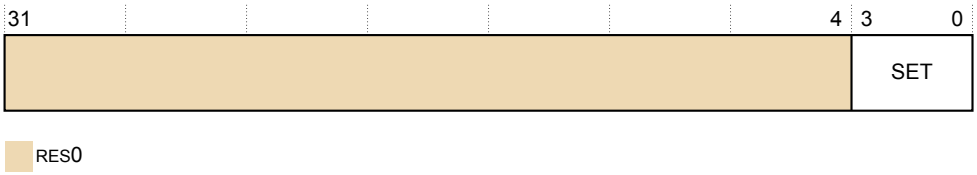

Figure D10-14 TRCCLAIMSET bit assignments

RES0, [31:4]

RES0 Reserved.

SET, [3:0]

On reads, for each bit:

`0` Claim tag bit is not implemented.

`1` Claim tag bit is implemented.

On writes, for each bit:

`0` Has no effect.

`1` Sets the relevant bit of the claim tag.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The TRCCLAIMSET can be accessed through the external debug interface, offset `0xFA0`.

D10.16 TRCCNTCTLR0, Counter Control Register 0

The TRCCNTCTLR0 controls the counter.

Bit field descriptions

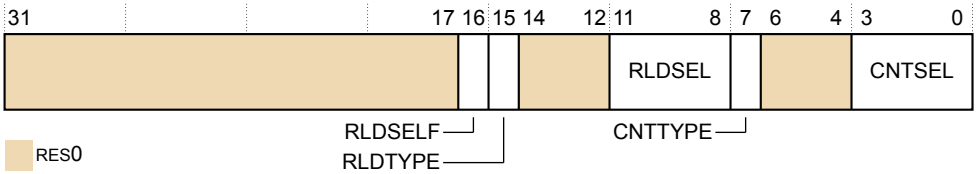

Figure D10-15 TRCCNTCTLR0 bit assignments

RES0, [31:17]

RES0 Reserved.

RLDSELF, [16]

Defines whether the counter reloads when it reaches zero:

0 The counter does not reload when it reaches zero. The counter only reloads based on RLDTYPE and RLDSEL.

1 The counter reloads when it reaches zero and the resource selected by CNTTYPE and CNTSEL is also active. The counter also reloads based on RLDTYPE and RLDSEL.

RLDTYPE, [15]

Selects the resource type for the reload:

0 Single selected resource.

1 Boolean combined resource pair.

RES0, [14:12]

RES0 Reserved.

RLDSEL, [11:8]

Selects the resource number, based on the value of RLDTYPE:

When RLDTYPE is 0, selects a single selected resource from 0-15 defined by bits[3:0].

When RLDTYPE is 1, selects a Boolean combined resource pair from 0-7 defined by bits[2:0].

CNTTYPE, [7]

Selects the resource type for the counter:

0 Single selected resource.

1 Boolean combined resource pair.

RES0, [6:4]

RES0 Reserved.

CNTSEL, [3:0]

Selects the resource number, based on the value of CNTTYPE:

When CNTTYPE is 0, selects a single selected resource from 0-15 defined by bits[3:0].

When CNTTYPE is 1, selects a Boolean combined resource pair from 0-7 defined by bits[2:0].

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The TRCCNTCTLR0 can be accessed through the external debug interface, offset `0x150`.

D10.17 TRCCNTCTLR1, Counter Control Register 1

The TRCCNTCTLR1 controls the counter.

Bit field descriptions

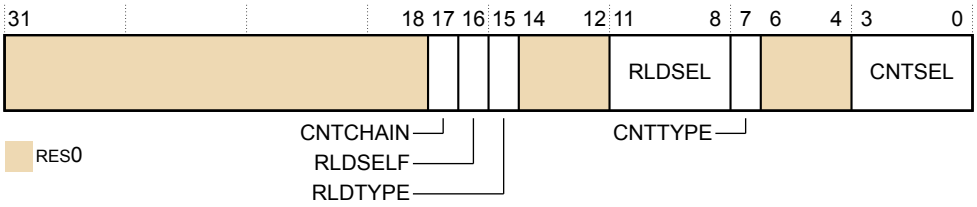

Figure D10-16 TRCCNTCTLR1 bit assignments

RES0, [31:18]

RES0 Reserved.

CNTCHAIN, [17]

Defines whether the counter decrements when the counter reloads. This enables two counters to be used in combination to provide a larger counter:

0 The counter operates independently from the counter. The counter only decrements based on CNTTYPE and CNTSEL.

1 The counter decrements when the counter reloads. The counter also decrements when the resource selected by CNTTYPE and CNTSEL is active.

RLDSELF, [16]

Defines whether the counter reloads when it reaches zero:

0 The counter does not reload when it reaches zero. The counter only reloads based on RLDTYPE and RLDSEL.

1 The counter reloads when it is zero and the resource selected by CNTTYPE and CNTSEL is also active. The counter also reloads based on RLDTYPE and RLDSEL.

RLDTYPE, [15]

Selects the resource type for the reload:

0 Single selected resource.

1 Boolean combined resource pair.

RES0, [14:12]

RES0 Reserved.

RLDSEL, [11:8]

Selects the resource number, based on the value of RLDTYPE:

When RLDTYPE is 0, selects a single selected resource from 0-15 defined by bits[3:0].

When RLDTYPE is 1, selects a Boolean combined resource pair from 0-7 defined by bits[2:0].

CNTTYPE, [7]

Selects the resource type for the counter:

0 Single selected resource.

1 Boolean combined resource pair.

RES0, [6:4]

RES0 Reserved.

CNTSEL, [3:0]

Selects the resource number, based on the value of CNTTYPE:

When CNTTYPE is 0, selects a single selected resource from 0-15 defined by bits[3:0].

When CNTTYPE is 1, selects a Boolean combined resource pair from 0-7 defined by bits[2:0].

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The TRCCNTCTLR1 can be accessed through the external debug interface, offset `0x154`.

D10.18 TRCCNTRLDVRn, Counter Reload Value Registers 0-1

The TRCCNTRLDVRn define the reload value for the counter.

Bit field descriptions

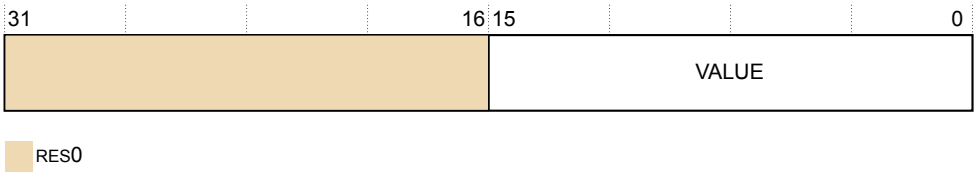

Figure D10-17 TRCCNTRLDVRn bit assignments

RES0, [31:16]

RES0 Reserved.

VALUE, [15:0]

Defines the reload value for the counter. This value is loaded into the counter each time the reload event occurs.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The TRCCNTRLDVRn registers can be accessed through the external debug interface, offsets:

TRCCNTRLDVR0

`0x140`.

TRCCNTRLDVR1

`0x144`.

D10.19 TRCCNTVRn, Counter Value Registers 0-1

The TRCCNTRLDVRn contain the current counter value.

Bit field descriptions

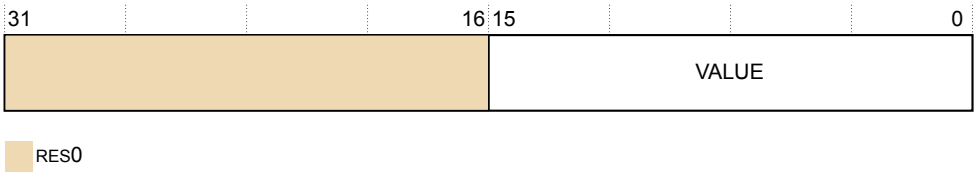

Figure D10-18 TRCCNTVRn bit assignments

RES0, [31:16]

RES0 Reserved.

VALUE, [15:0]

Contains the current counter value.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The TRCCNTRLDVRn registers can be accessed through the external debug interface, offsets:

TRCCNTVR0

`0x160`.

TRCCNTVR1

`0x164`.

D10.20 TRCCONFIGR, Trace Configuration Register

The TRCCONFIGR controls the tracing options.

Bit field descriptions

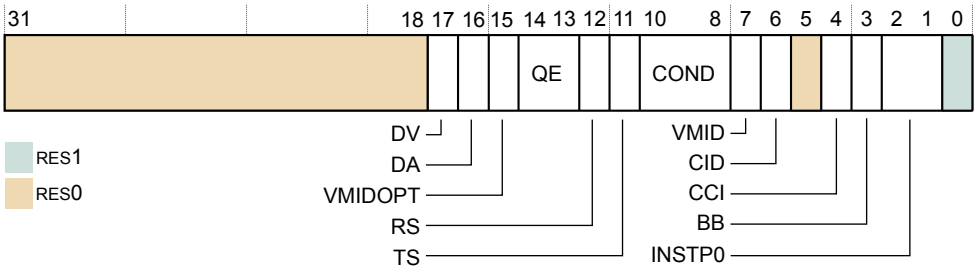

Figure D10-19 TRCCONFIGR bit assignments

RES0, [31:18]

RES0 Reserved.

DV, [17]

Enables data value tracing. The possible values are:

0 Disables data value tracing.

1 Enables data value tracing.

DA, [16]

Enables data address tracing. The possible values are:

0 Disables data address tracing.

1 Enables data address tracing.

VMIDOPT, [15]

Configures the Virtual context identifier value used by the trace unit, both for trace generation and in the Virtual context identifier comparators. The possible values are:

0b0 VTTBR_EL2.VMID is used. If the trace unit supports a Virtual context identifier larger than the VTTBR_EL2.VMID, the upper unused bits are always zero. If the trace unit supports a Virtual context identifier larger than 8 bits and if the VTCR_EL2.VS bit forces use of an 8-bit Virtual context identifier, bits [15:8] of the trace unit Virtual context identifier are always zero.

0b1 CONTEXTIDR_EL2 is used. TRCIDR2.VMIDOPT indicates whether this field is implemented.

QE, [14:13]

Enables Q element. The possible values are:

0b00 Q elements are disabled.

0b01 Q elements with instruction counts are disabled. Q elements without instruction counts are disabled.

0b10 Reserved.

0b11 Q elements with and without instruction counts are enabled.

RS, [12]

Enables the return stack. The possible values are:

0 Disables the return stack.

`1` Enables the return stack.

TS, [11]

Enables global timestamp tracing. The possible values are:

`0` Disables global timestamp tracing.

`1` Enables global timestamp tracing.

COND, [10:8]

Enables conditional instruction tracing. The possible values are:

`0b000` Conditional instruction tracing is disabled.

`0b001` Conditional load instructions are traced.

`0b010` Conditional store instructions are traced.

`0b011` Conditional load and store instructions are traced.

`0b111` All conditional instructions are traced.

VMID, [7]

Enables VMID tracing. The possible values are:

`0` Disables VMID tracing.

`1` Enables VMID tracing.

CID, [6]

Enables context ID tracing. The possible values are:

`0` Disables context ID tracing.

`1` Enables context ID tracing.

RES0, [5]

RES0 Reserved.

CCI, [4]

Enables cycle counting instruction trace. The possible values are:

`0` Disables cycle counting instruction trace.

`1` Enables cycle counting instruction trace.

BB, [3]

Enables branch broadcast mode. The possible values are:

`0` Disables branch broadcast mode.

`1` Enables branch broadcast mode.

INSTP0, [2:1]

Controls whether load and store instructions are traced as P0 instructions. The possible values are:

`0b00` Load and store instructions are not traced as P0 instructions.

`0b01` Load instructions are traced as P0 instructions.

`0b10` Store instructions are traced as P0 instructions.

`0b11` Load and store instructions are traced as P0 instructions.

RES1, [0]

RES1 Reserved.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The TRCCONFIGR can be accessed through the external debug interface, offset `0x010`.

D10.21 TRCDEVAFF0, Device Affinity Register 0

The TRCDEVAFF0 provides an additional core identification mechanism for scheduling purposes in a cluster. TRCDEVAFF0 is a read-only copy of MPIDR accessible from the external debug interface.

Bit field descriptions

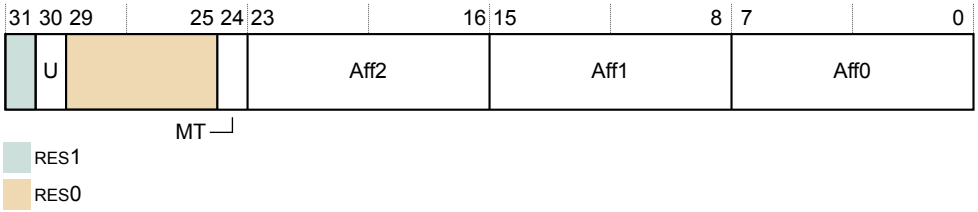

Figure D10-20 TRCDEVAFF0 bit assignments

RES1, [31]

RES1 Reserved.

U, [30]

Indicates a single core system, as distinct from core 0 in a cluster. This value is:

`0` Core is part of a multiprocessor system. This is the value for implementations with more than one core, and for implementations with an ACE or CHI master interface.

`1` Core is part of a uniprocessor system. This is the value for single core implementations with an AXI master interface.

RES0, [29:25]

RES0 Reserved.

MT, [24]

Indicates whether the lowest level of affinity consists of logical cores that are implemented using a multi-threading type approach. This value is:

`0` Performance of cores at the lowest affinity level is largely independent.

Aff2, [23:16]

Affinity level 2. Second highest level affinity field.

Indicates the value read in the **CLUSTERIDAFF2** configuration signal.

Aff1, [15:8]

Affinity level 1. Third highest level affinity field.

Indicates the value read in the **CLUSTERIDAFF1** configuration signal.

Aff0, [7:0]

Affinity level 0. Lowest level affinity field.

Indicates the core number in the Cortex-A75 core. The possible values are:

`0x0`	A cluster with one core only.
`0x0`, `0x1`	A cluster with two cores.
`0x0`, `0x1`, `0x2`	A cluster with three cores.
`0x0`, `0x1`, `0x2`, `0x3`	A cluster with four cores.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The TRCDEVAFF0 can be accessed through the external debug interface, offset `0xFA8`.

D10.22 TRCDEVAFF1, Device Affinity Register 1

The TRCDEVAFF1 is a read-only copy of MPIDR_EL1[63:32] as seen from EL3, unaffected by VMPIDR_EL2.

D10.23 TRCDEVARCH, Device Architecture Register

The TRCDEVARCH identifies the ETM trace unit as an ETMv4 component.

Bit field descriptions

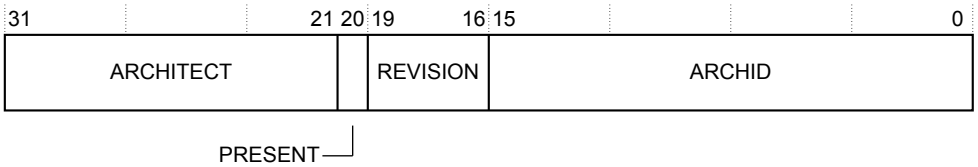

Figure D10-21 TRCDEVARCH bit assignments

ARCHITECT, [31:21]
Defines the architect of the component:

`0x4` Arm JEP continuation.

`0x3B` Arm JEP 106 code.

PRESENT, [20]
Indicates the presence of this register:

`0b1` Register is present.

REVISION, [19:16]
Architecture revision:

`0x02` Architecture revision 2.

ARCHID, [15:0]
Architecture ID:

`0x4A13` ETMv4 component.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The TRCDEVARCH can be accessed through the external debug interface, offset `0xFBC`.

D10.24 TRCDEVID, Device ID Register

The TRCDEVID indicates the capabilities of the ETM trace unit.

Bit field descriptions

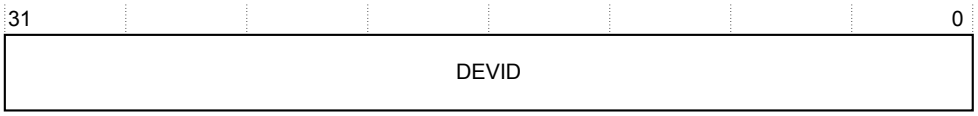

Figure D10-22 TRCDEVID bit assignments

DEVID, [31:0]

RAZ. There are no component-defined capabilities.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The TRCDEVID can be accessed through the external debug interface, offset `0xFC8`.

D10.25 TRCDEVTYPE, Device Type Register

The TRCDEVTYPE indicates the type of the component.

Bit field descriptions

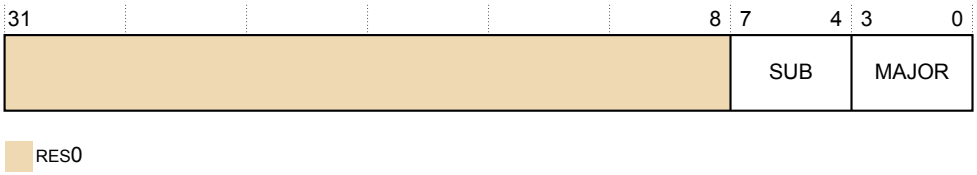

Figure D10-23 TRCDEVTYPE bit assignments

RES0, [31:8]

RES0 Reserved.

SUB, [7:4]

The sub-type of the component:

`0b0001` Core trace.

MAJOR, [3:0]

The main type of the component:

`0b0011` Trace source.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The TRCDEVTYPE can be accessed through the external debug interface, offset `0xFCC`.

D10.26 TRCEVENTCTL0R, Event Control 0 Register

The TRCEVENTCTL0R controls the tracing of events in the trace stream. The events also drive the external outputs from the ETM trace unit. The events are selected from the Resource Selectors.

Bit field descriptions

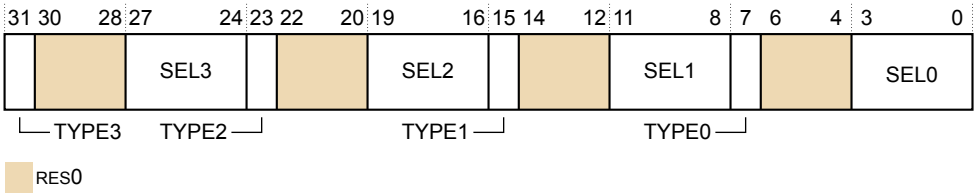

Figure D10-24 TRCEVENTCL0R bit assignments

TYPE3, [31]

Selects the resource type for trace event 3:

0 Single selected resource.

1 Boolean combined resource pair.

RES0, [30:28]

RES0 Reserved.

SEL3, [27:24]

Selects the resource number, based on the value of TYPE3:

When TYPE3 is 0, selects a single selected resource from 0-15 defined by bits[3:0].

When TYPE3 is 1, selects a Boolean combined resource pair from 0-7 defined by bits[2:0].

TYPE2, [23]

Selects the resource type for trace event 2:

0 Single selected resource.

1 Boolean combined resource pair.

RES0, [22:20]

RES0 Reserved.

SEL2, [19:16]

Selects the resource number, based on the value of TYPE2:

When TYPE2 is 0, selects a single selected resource from 0-15 defined by bits[3:0].

When TYPE2 is 1, selects a Boolean combined resource pair from 0-7 defined by bits[2:0].

TYPE1, [15]

Selects the resource type for trace event 1:

0 Single selected resource.

1 Boolean combined resource pair.

RES0, [14:12]

RES0 Reserved.

SEL1, [11:8]

Selects the resource number, based on the value of TYPE1:

When TYPE1 is 0, selects a single selected resource from 0-15 defined by bits[3:0].

When TYPE1 is 1, selects a Boolean combined resource pair from 0-7 defined by bits[2:0].

TYPE0, [7]

Selects the resource type for trace event 0:

`0` Single selected resource.

`1` Boolean combined resource pair.

RES0, [6:4]

RES0 Reserved.

SEL0, [3:0]

Selects the resource number, based on the value of TYPE0:

When TYPE0 is 0, selects a single selected resource from 0-15 defined by bits[3:0].

When TYPE0 is 1, selects a Boolean combined resource pair from 0-7 defined by bits[2:0].

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The TRCEVENTCTL0R can be accessed through the external debug interface, offset `0x020`.

D10.27 TRCEVENTCL1R, Event Control 1 Register

The TRCEVENTCTL1R controls the behavior of the events that TRCEVENTCTL0R selects.

Bit field descriptions

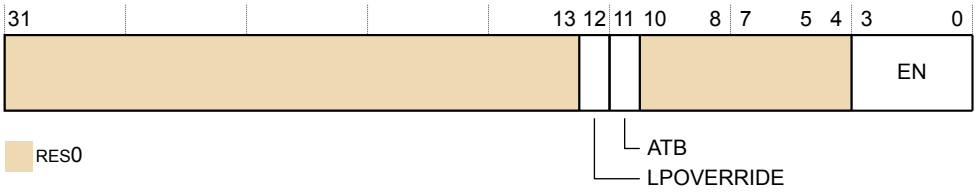

Figure D10-25 TRCEVENTCL1R bit assignments

RES0, [31:13]

RES0 Reserved.

LPOVERRIDE, [12]

Low-power state behavior override:

`0` Low-power state behavior unaffected.

`1` Low-power state behavior overridden. The resources and Event trace generation are unaffected by entry to a low-power state.

ATB, [11]

ATB trigger enable:

`0` ATB trigger disabled.

`1` ATB trigger enabled.

RES0, [10:4]

RES0 Reserved.

EN, [3:0]

One bit per event, to enable generation of an event element in the instruction trace stream when the selected event occurs:

`0` Event does not cause an event element.

`1` Event causes an event element.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The TRCEVENTCTL1R can be accessed through the external debug interface, offset `0x024`.

D10.28 TRCEXTINSELR, External Input Select Register

The TRCEXTINSELR controls the selectors that choose an external input as a resource in the ETM trace unit. You can use the Resource Selectors to access these external input resources.

Bit field descriptions

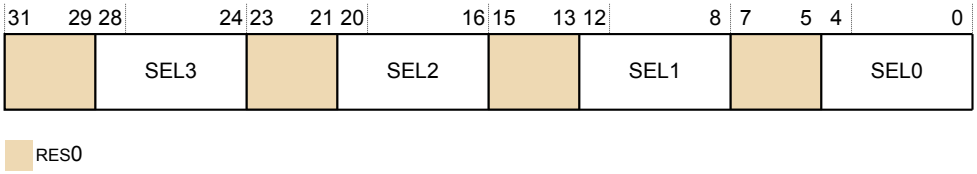

Figure D10-26 TRCEXTINSELR bit assignments

RES0, [31:29]

RES0 Reserved.

SEL3, [28:24]

Selects an event from the external input bus for External Input Resource 3.

RES0, [23:21]

RES0 Reserved.

SEL2, [20:16]

Selects an event from the external input bus for External Input Resource 2.

RES0, [15:13]

RES0 Reserved.

SEL1, [12:8]

Selects an event from the external input bus for External Input Resource 1.

RES0, [7:5]

RES0 Reserved.

SEL0, [4:0]

Selects an event from the external input bus for External Input Resource 0.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The TRCEXTINSELR can be accessed through the external debug interface, offset `0x120`.

D10.29 TRCIDR0, ID Register 0

The TRCIDR0 returns the tracing capabilities of the ETM trace unit.

Bit field descriptions

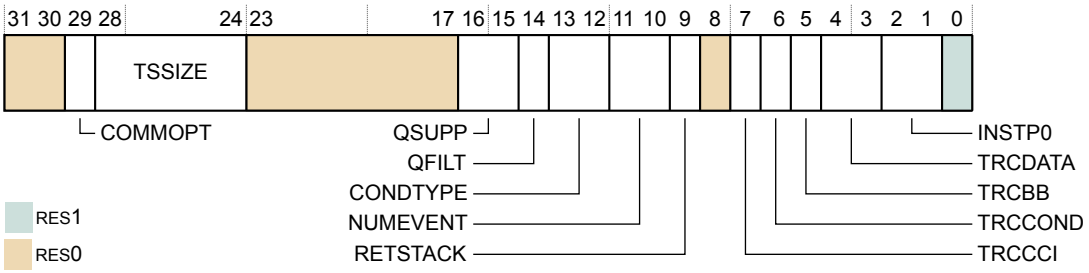

Figure D10-27 TRCIDR0 bit assignments

RES0, [31:30]

RES0 Reserved.

COMMOPT, [29]

Indicates the meaning of the commit field in some packets:

`1` Commit mode 1.

TSSIZE, [28:24]

Global timestamp size field:

`0b01000` Implementation supports a maximum global timestamp of 64 bits.

RES0, [23:17]

RES0 Reserved.

QSUPP, [16:15]

Indicates Q element support:

`0b00` Q elements not supported.

QFILT, [14]

Indicates Q element filtering support:

`0b0` Q element filtering not supported.

CONDTYPE, [13:12]

Indicates how conditional results are traced:

`0b00` Conditional trace not supported.

NUMEVENT, [11:10]

Number of events supported in the trace, minus 1:

`0b11` Four events supported.

RETSTACK, [9]

Return stack support:

`1` Return stack implemented.

RES0, [8]

RES0 Reserved.

TRCCCI, [7]

Support for cycle counting in the instruction trace:

`1` Cycle counting in the instruction trace is implemented.

TRCCOND, [6]

Support for conditional instruction tracing:

`0` Conditional instruction tracing is not supported.

TRCBB, [5]

Support for branch broadcast tracing:

`1` Branch broadcast tracing is implemented.

TRCDATA, [4:3]

Conditional tracing field:

`0b00` Tracing of data addresses and data values is not implemented.

INSTP0, [2:1]

P0 tracing support field:

`0b00` Tracing of load and store instructions as P0 elements is not supported.

RES1, [0]

RES1 Reserved.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The TRCIDR0 can be accessed through the external debug interface, offset `0x1E0`.

D10.30 TRCIDR1, ID Register 1

The TRCIDR1 returns the base architecture of the trace unit.

Bit field descriptions

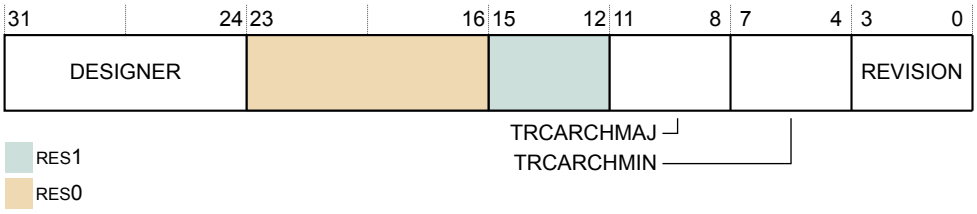

Figure D10-28 TRCIDR1 bit assignments

DESIGNER, [31:24]
Indicates which company designed the trace unit:

`0x41` Arm.

RES0, [23:16]

RES0 Reserved.

RES1, [15:12]

RES1 Reserved.

TRCARCHMAJ, [11:8]
Major trace unit architecture version number:

`0b0100` ETMv4.

TRCARCHMIN, [7:4]
Minor trace unit architecture version number:

`0x2` ETMv4.2

REVISION, [3:0]
Trace unit implementation revision number:

1 ETM revision.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The TRCIDR1 can be accessed through the external debug interface, offset `0x1E4`.

D10.31 TRCIDR2, ID Register 2

The TRCIDR2 returns the maximum size of six parameters in the trace unit.

The parameters are:

- Cycle counter.
- Data value.
- Data address.
- VMID.
- Context ID.
- Instruction address.

Bit field descriptions

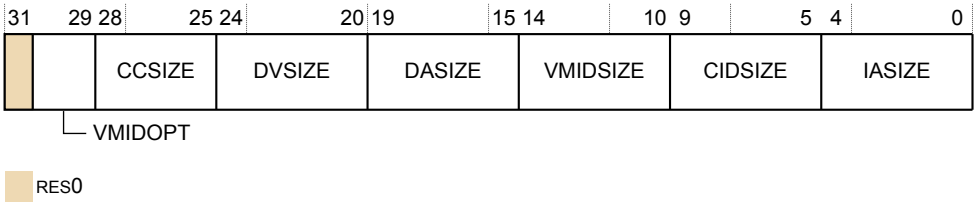

Figure D10-29 TRCIDR2 bit assignments

RES0, [31]

RES0 Reserved.

VMIDOPT, [30:29]

Indicates the options for observing the Virtual context identifier:

`0x1` VMIDOPT is implemented.

CCSIZE, [28:25]

Size of the cycle counter in bits minus 12:

`0x0` The cycle counter is 12 bits in length.

DVSIZE, [24:20]

Data value size in bytes:

`0x00` Data value tracing is not implemented.

DASIZE, [19:15]

Data address size in bytes:

`0x00` Data address tracing is not implemented.

VMIDSIZE, [14:10]

Virtual Machine ID size:

`0x4` Maximum of 32-bit Virtual Machine ID size.

CIDSIZE, [9:5]

Context ID size in bytes:

`0x4` Maximum of 32-bit Context ID size.

IASIZE, [4:0]

Instruction address size in bytes:

`0x8` Maximum of 64-bit address size.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The TRCIDR2 can be accessed through the external debug interface, offset `0x1E8`.

D10.32 TRCIDR3, ID Register 3

The TRCIDR3 indicates:

- Whether TRCVICTLR is supported.
- The number of cores available for tracing.
- If an exception level supports instruction tracing.
- The minimum threshold value for instruction trace cycle counting.
- Whether the synchronization period is fixed.
- Whether TRCSTALLCTLR is supported and if so whether it supports trace overflow prevention and supports stall control of the core.

Bit field descriptions

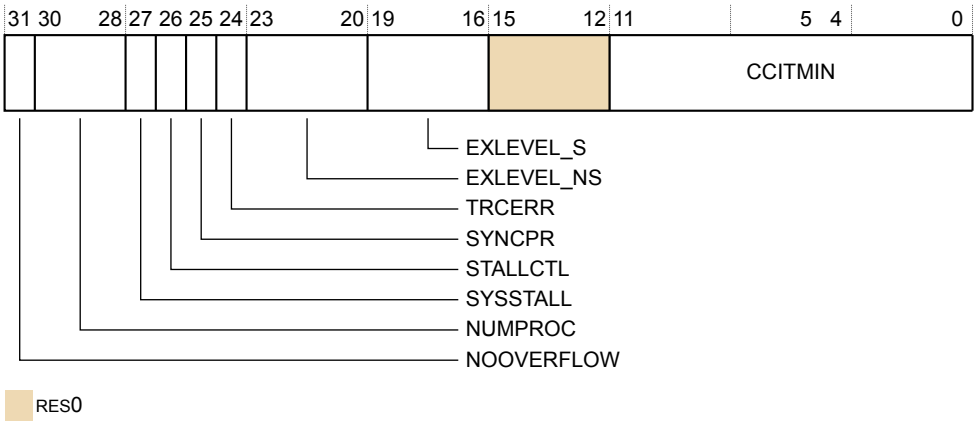

Figure D10-30 TRCIDR3 bit assignments

NOOVERFLOW, [31]

Indicates whether TRCSTALLCTLR.NOOVERFLOW is implemented:

`0` TRCSTALLCTLR.NOOVERFLOW is not implemented.

NUMPROC, [30:28]

Indicates the number of cores available for tracing:

`0b000` The trace unit can trace one core, ETM trace unit sharing not supported.

SYSSTALL, [27]

Indicates whether stall control is implemented:

`1` The system supports core stall control.

STALLCTL, [26]

Indicates whether TRCSTALLCTLR is implemented:

`1` TRCSTALLCTLR is implemented.

This field is used in conjunction with SYSSTALL.

SYNCPR, [25]

Indicates whether there is a fixed synchronization period:

`0` TRCSYNCPR is read-write so software can change the synchronization period.

TRCERR, [24]

Indicates whether TRCVICTLR.TRCERR is implemented:

`1` TRCVICTLR.TRCERR is implemented.

EXLEVEL_NS, [23:20]

Each bit controls whether instruction tracing in Non-secure state is implemented for the corresponding Exception level:

`0b0111` Instruction tracing is implemented for Non-secure EL0, EL1, and EL2 Exception levels.

EXLEVEL_S, [19:16]

Each bit controls whether instruction tracing in Secure state is implemented for the corresponding Exception level:

`0b1011` Instruction tracing is implemented for Secure EL0, EL1, and EL3 Exception levels.

RES0, [15:12]

RES0 Reserved.

CCITMIN, [11:0]

The minimum value that can be programmed in TRCCCCTLR.THRESHOLD:

`0x004` Instruction trace cycle counting minimum threshold is 4.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The TRCIDR3 can be accessed through the external debug interface, offset `0x1EC`.

D10.33 TRCIDR4, ID Register 4

The TRCIDR4 indicates the resources available in the ETM trace unit.

Bit field descriptions

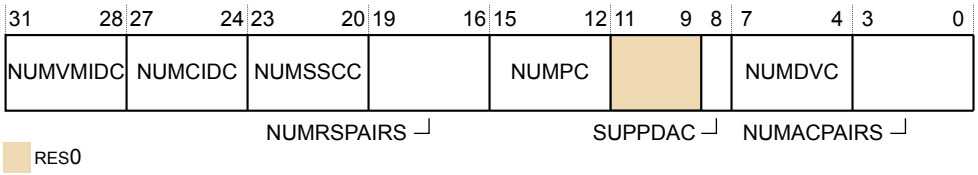

Figure D10-31 TRCIDR4 bit assignments

NUMVMIDC, [31:28]
Indicates the number of VMID comparators available for tracing:

`0x1` One VMID comparator is available.

NUMCIDC, [27:24]
Indicates the number of CID comparators available for tracing:

`0x1` One Context ID comparator is available.

NUMSSCC, [23:20]
Indicates the number of single-shot comparator controls available for tracing:

`0x1` One single-shot comparator control is available.

NUMRSPAIRS, [19:16]
Indicates the number of resource selection pairs available for tracing:

`0x7` Eight resource selection pairs are available.

NUMPC, [15:12]
Indicates the number of core comparator inputs available for tracing:

`0x0` Core comparator inputs are not implemented.

RES0, [11:9]

RES0 Reserved.

SUPPDAC, [8]
Indicates whether the implementation supports data address comparisons: This value is:

`0` Data address comparisons are not implemented.

NUMDVC, [7:4]
Indicates the number of data value comparators available for tracing:

`0x0` Data value comparators not implemented.

NUMACPAIRS, [3:0]
Indicates the number of address comparator pairs available for tracing:

`0x4` Four address comparator pairs are implemented.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The TRCIDR4 can be accessed through the external debug interface, offset `0x1F0`.

D10.34 TRCIDR5, ID Register 5

The TRCIDR5 returns how many resources the trace unit supports.

Bit field descriptions

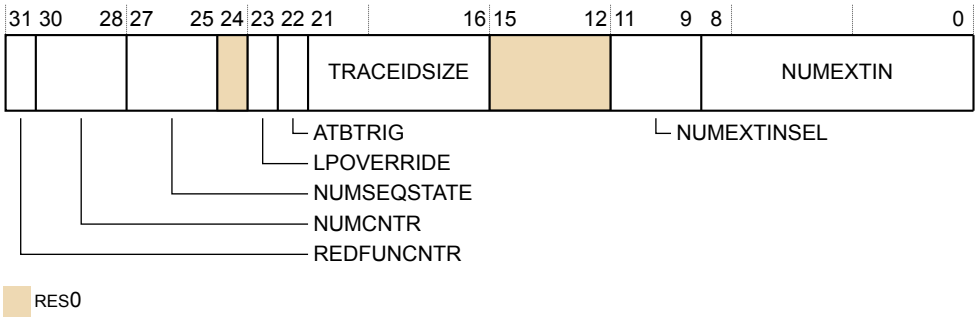

Figure D10-32 TRCIDR5 bit assignments

REDFUNCNTR, [31]
Reduced Function Counter implemented:

`0` Reduced Function Counter not implemented.

NUMCNTR, [30:28]
Number of counters implemented:

`0b010` Two counters implemented.

NUMSEQSTATE, [27:25]
Number of sequencer states implemented:

`0b100` Four sequencer states implemented.

RES0, [24]

RES0 Reserved.

LPOVERRIDE, [23]
Low-power state override support:

`1` Low-power state override support implemented.

ATBTRIG, [22]
ATB trigger support:

`1` ATB trigger support implemented.

TRACEIDSIZE, [21:16]
Number of bits of trace ID:

`0x07` Seven-bit trace ID implemented.

RES0, [15:12]

RES0 Reserved.

NUMEXTINSEL, [11:9]
Number of external input selectors implemented:

`0b100` Four external input selectors implemented.

NUMEXTIN, [8:0]
Number of external inputs implemented:

`0x20` 32 external inputs implemented.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The TRCIDR5 can be accessed through the external debug interface, offset `0x1F4`.

D10.35 TRCIDR8, ID Register 8

The TRCIDR8 returns the maximum speculation depth of the instruction trace stream.

Bit field descriptions

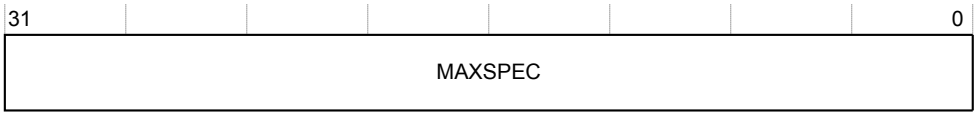

Figure D10-33 TRCIDR8 bit assignments

MAXSPEC, [31:0]

The maximum number of P0 elements in the trace stream that can be speculative at any time.

`0` Maximum speculation depth of the instruction trace stream.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The TRCIDR8 can be accessed through the external debug interface, offset `0x180`.

D10.36 TRCIDR9, ID Register 9

The TRCIDR9 returns the number of P0 right-hand keys that the trace unit can use.

Bit field descriptions

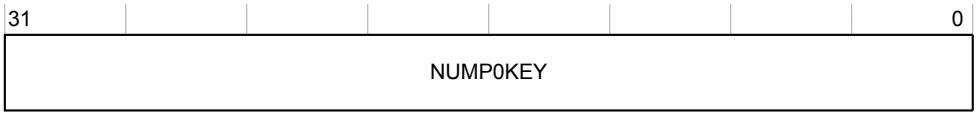

Figure D10-34 TRCID9 bit assignments

NUMP0KEY, [31:0]

The number of P0 right-hand keys that the trace unit can use.

`0` Number of P0 right-hand keys.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The TRCIDR9 can be accessed through the external debug interface, offset `0x184`.

D10.37 TRCIDR10, ID Register 10

The TRCIDR10 returns the number of P1 right-hand keys that the trace unit can use.

Bit field descriptions

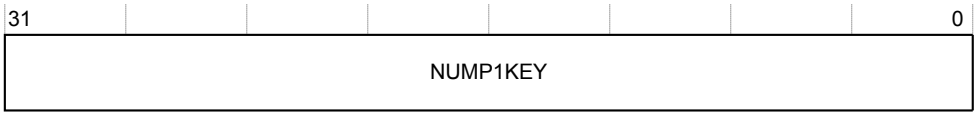

Figure D10-35 TRCIDR10 bit assignments

NUMP1KEY, [31:0]

The number of P1 right-hand keys that the trace unit can use.

`0` Number of P1 right-hand keys.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The TRCIDR10 can be accessed through the external debug interface, offset `0x188`.

D10.38 TRCIDR11, ID Register 11

The TRCIDR11 returns the number of special P1 right-hand keys that the trace unit can use.

Bit field descriptions

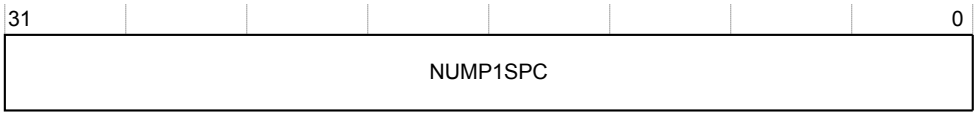

Figure D10-36 TRCIDR11 bit assignments

NUMP1SPC, [31:0]

The number of special P1 right-hand keys that the trace unit can use.

`0` Number of special P1 right-hand keys.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The TRCIDR11 can be accessed through the external debug interface, offset `0x18C`.

D10.39 TRCIDR12, ID Register 12

The TRCIDR12 returns the number of conditional instruction right-hand keys that the trace unit can use.

Bit field descriptions

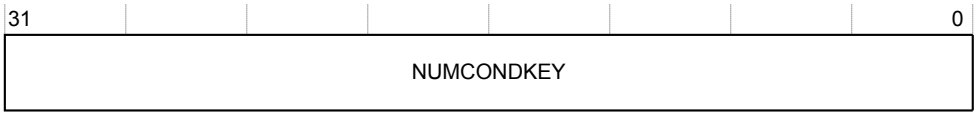

Figure D10-37 TRCIDR12 bit assignments

NUMCONDKEY, [31:0]

The number of conditional instruction right-hand keys that the trace unit can use, including normal and special keys.

`0` Number of conditional instruction right-hand keys.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The TRCIDR12 can be accessed through the external debug interface, offset `0x190`.

D10.40 TRCIDR13, ID Register 13

The TRCIDR13 returns the number of special conditional instruction right-hand keys that the trace unit can use.

Bit field descriptions

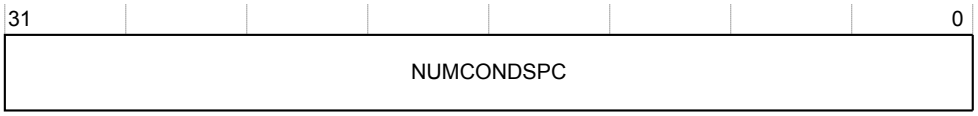

Figure D10-38 TRCIDR13 bit assignments

NUMCONDSPC, [31:0]

The number of special conditional instruction right-hand keys that the trace unit can use, including normal and special keys.

`0` Number of special conditional instruction right-hand keys.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The TRCIDR13 can be accessed through the external debug interface, offset `0x194`.

D10.41 TRCIMSPEC0, Implementation Specific Register 0

The TRCIMSPEC0 shows the presence of any implementation specific features, and enables any features that are provided.

Bit field descriptions

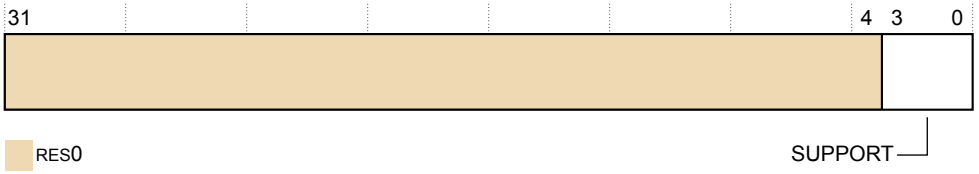

Figure D10-39 TRCIMSPEC0 bit assignments

RES0, [31:4]

RES0 Reserved.

SUPPORT, [3:0]

`0` No implementation specific extensions are supported.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The TRCIMSPEC0 can be accessed through the external debug interface, offset `0x1C0`.

D10.42 TRCITATBIDR, Integration ATB Identification Register

The TRCITATBIDR sets the state of output pins mentioned in the bit descriptions in this section.

Bit field descriptions

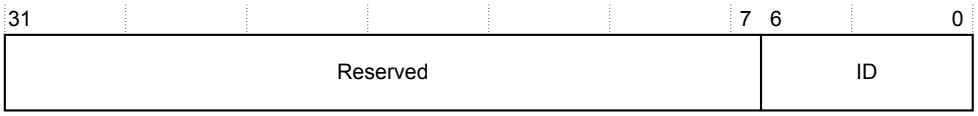

Figure D10-40 TRCITATBIDR bit assignments

[31:7]

Reserved. Read undefined.

ID, [6:0]

Drives the **ATIDMn[6:0]** output pins.

When a bit is set to 0, the corresponding output pin is LOW.

When a bit is set to 1, the corresponding output pin is HIGH.

The TRCITATBIDR bit values correspond to the physical state of the output pins.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The TRCITATBIDR can be accessed through the external debug interface, offset `0xEE4`.

D10.43 TRCITCTRL, Integration Mode Control Register

The TRCITCTRL enables topology detection or integration testing, by putting the ETM trace unit into integration mode.

Bit field descriptions

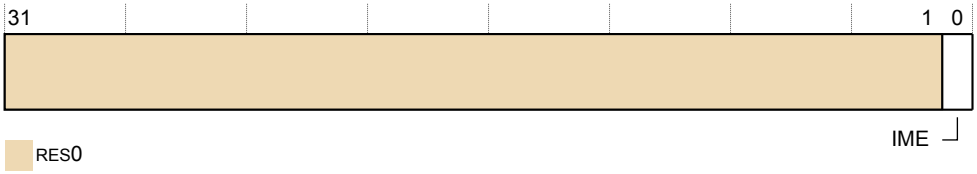

Figure D10-41 TRCITCTRL bit assignments

RES0, [31:1]

RES0 Reserved.

IME, [0]

Integration mode enable bit. The possible values are:

`0` The trace unit is not in integration mode.

`1` The trace unit is in integration mode. This mode enables:

- A debug agent to perform topology detection.
- SoC test software to perform integration testing.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The TRCITCTRL can be accessed through the external debug interface, offset `0xF00`.

D10.44 TRCITIATBINR, Integration Instruction ATB In Register

The TRCITIATBINR reads the state of the input pins described in this section.

Bit field descriptions

Figure D10-42 TRCITIATBINR bit assignments

For all non-reserved bits:

- When an input pin is LOW, the corresponding register bit is 0.
- When an input pin is HIGH, the corresponding register bit is 1.
- The TRCITIATBINR bit values always correspond to the physical state of the input pins.

[31:2]
Reserved. Read undefined.

AFVALIDM, [1]
Returns the value of the **AFVALIDMn** input pin.

ATREADYM, [0]
Returns the value of the **ATREADYMn** input pin.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The TRCITIATBINR can be accessed through the external debug interface, offset `0xEF4`.

D10.45 TRCITIATBOUTR, Integration Instruction ATB Out Register

The TRCITIATBOUTR sets the state of the output pins mentioned in the bit descriptions in this section.

Bit field descriptions

Figure D10-43 TRCITIATBOUTR bit assignments

For all non-reserved bits:

- When a bit is set to 0, the corresponding output pin is LOW.
- When a bit is set to 1, the corresponding output pin is HIGH.
- The TRCITIATBOUTR bit values always correspond to the physical state of the output pins.

[31:10]
Reserved. Read undefined.

BYTES, [9:8]
Drives the **ATBYTESMn[1:0]** output pins.

[7:2]
Reserved. Read undefined.

AFREADY, [1]
Drives the **AFREADYMn** output pin.

ATVALID, [0]
Drives the **ATVALIDMn** output pin.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The TRCITIATBOUTR can be accessed through the external debug interface, offset `0xEFC`.

D10.46 TRCITIDATAR, Integration Instruction ATB Data Register

The TRCITIDATAR sets the state of the **ATDATAMn** output pins shown in the TRCITIDATAR bit assignments table.

Bit field descriptions

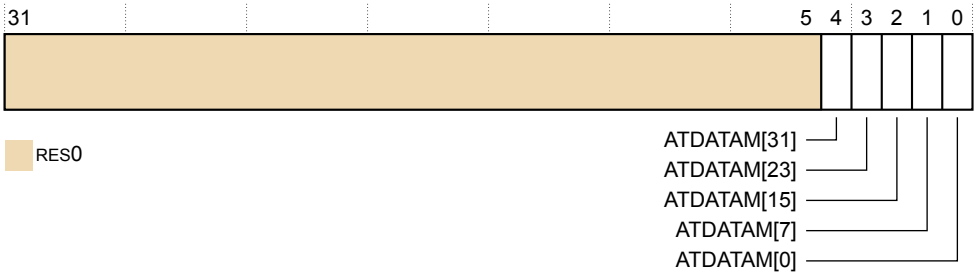

Figure D10-44 TRCITIDATAR bit assignments

RES0, [31:5]

RES0 Reserved.

ATDATAM[31], [4]
Drives the ATDATAM[31] output.[o]

ATDATAM[23], [3]
Drives the ATDATAM[23] output.[o]

ATDATAM[15], [2]
Drives the ATDATAM[15] output.[o]

ATDATAM[7], [1]
Drives the ATDATAM[7] output.[o]

ATDATAM[0], [0]
Drives the ATDATAM[0] output.[o]

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The TRCITIDATAR can be accessed through the external debug interface, offset `0xEEC`.

[o] When a bit is set to `0`, the corresponding output pin is LOW. When a bit is set to `1`, the corresponding output pin is HIGH. The TRCITIDATAR bit values correspond to the physical state of the output pins.

D10.47 TRCLAR, Software Lock Access Register

The TRCLAR controls access to registers using the memory-mapped interface, when **PADDRDBG31** is LOW.

When the software lock is set, write accesses using the memory-mapped interface to all ETM trace unit registers are ignored, except for write accesses to the TRCLAR.

When the software lock is set, read accesses of TRCPDSR do not change the TRCPDSR.STICKYPD bit. Read accesses of all other registers are not affected.

Bit field descriptions

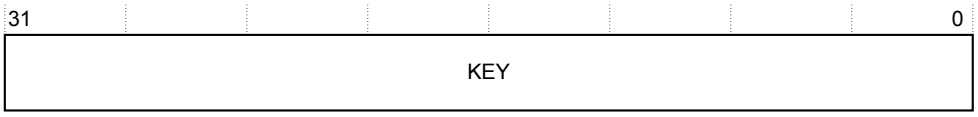

Figure D10-45 TRCLAR bit assignments

KEY, [31:0]

Software lock key value:

`0xC5ACCE55` Clear the software lock.

All other write values set the software lock.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The TRCLAR can be accessed through the external debug interface, offset `0xFB0`.

D10.48 TRCLSR, Software Lock Status Register

The TRCLSR determines whether the software lock is implemented, and indicates the current status of the software lock.

Bit field descriptions

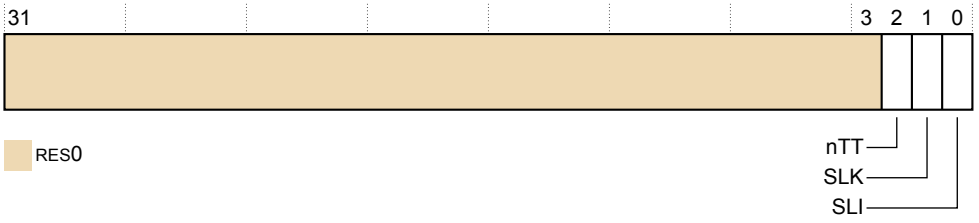

Figure D10-46 TRCLSR bit assignments

RES0, [31:3]

RES0 Reserved.

nTT, [2]

Indicates size of TRCLAR:

`0` TRCLAR is always 32 bits.

SLK, [1]

Software lock status:

`0` Software lock is clear.

`1` Software lock is set.

SLI, [0]

Indicates whether the software lock is implemented on this interface.

`1` Software lock is implemented on this interface.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The TRCLSR can be accessed through the external debug interface, offset `0xFB4`.

D10.49 TRCCNTVRn, Counter Value Registers 0-1

The TRCCNTVRn contains the current counter value.

Bit field descriptions

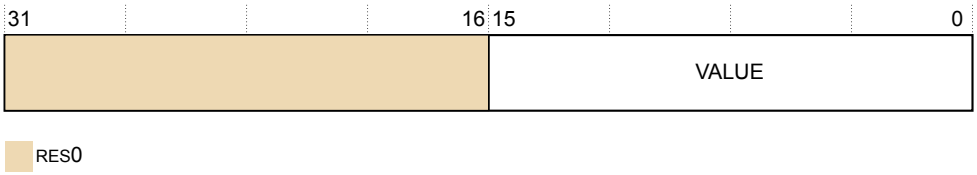

Figure D10-47 TRCCNTVRn bit assignments

RES0, [31:16]

RES0 Reserved.

VALUE, [15:0]

Contains the current counter value.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The TRCCNTVRn registers can be accessed through the external debug interface, offsets:

TRCCNTVR0

`0x160`.

TRCCNTVR1

`0x164`.

D10.50 TRCOSLAR, OS Lock Access Register

The TRCOSLAR sets and clears the OS Lock, to lock out external debugger accesses to the ETM trace unit registers.

Bit Field Assignments

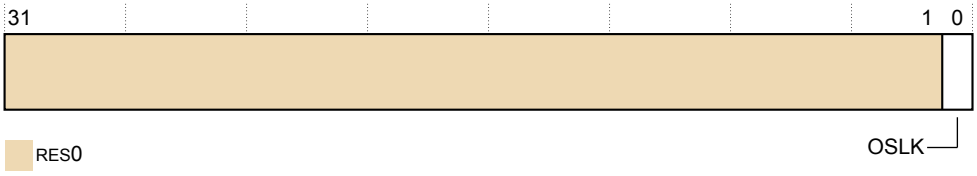

Figure D10-48 TRCOSLAR bit assignments

RES0, [31:1]

RES0 Reserved.

OSLK, [0]

OS Lock key value:

`0` Unlock the OS Lock.

`1` Lock the OS Lock.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The TRCOSLAR can be accessed through the external debug interface, offset `0x300`.

D10.51 TRCOSLSR, OS Lock Status Register

The TRCOSLSR returns the status of the OS Lock.

Bit field descriptions

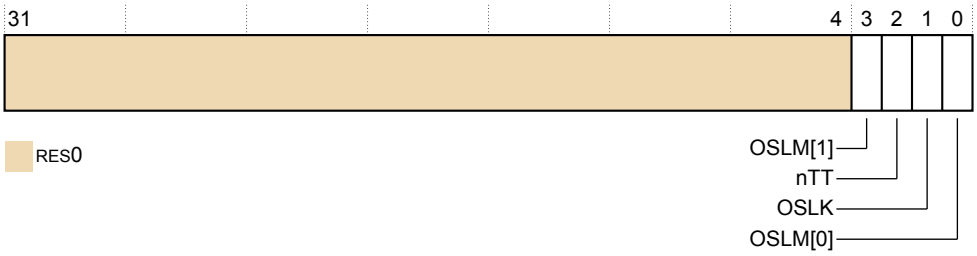

Figure D10-49 TRCOSLSR bit assignments

RES0, [31:4]

RES0 Reserved.

OSLM[1], [3]

OS Lock model [1] bit. This bit is combined with OSLM[0] to form a two-bit field that indicates the OS Lock model is implemented.

The value of this field is always `0b10`, indicating that the OS Lock is implemented.

nTT, [2]

This bit is RAZ, that indicates that software must perform a 32-bit write to update the TRCOSLAR.

OSLK, [1]

OS Lock status bit:

`0` OS Lock is unlocked.

`1` OS Lock is locked.

OSLM[0], [0]

OS Lock model [0] bit. This bit is combined with OSLM[1] to form a two-bit field that indicates the OS Lock model is implemented.

The value of this field is always `0b10`, indicating that the OS Lock is implemented.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The TRCOSLSR can be accessed through the external debug interface, offset `0x304`.

D10.52 TRCPDCR, Power Down Control Register

The TRCPDCR request to the system power controller to keep the ETM trace unit powered up.

Bit field descriptions

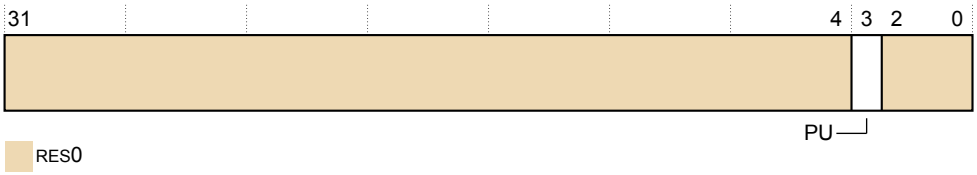

Figure D10-50 TRCPDCR bit assignments

RES0, [31:4]

RES0 Reserved.

PU, [3]

Powerup request, to request that power to the ETM trace unit and access to the trace registers is maintained:

`0` Power not requested.

`1` Power requested.

This bit is reset to 0 on a trace unit reset.

RES0, [2:0]

RES0 Reserved.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The TRCPDCR can be accessed through the external debug interface, offset `0x310`.

D10.53 TRCPDSR, Power Down Status Register

The TRCPDSR indicates the power down status of the ETM trace unit.

Bit field descriptions

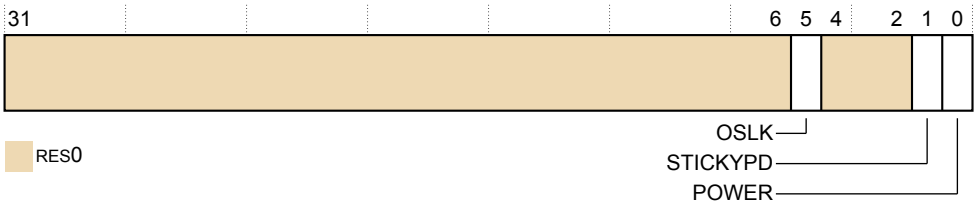

Figure D10-51 TRCPDSR bit assignments

RES0, [31:6]

RES0 Reserved.

OSLK, [5]

OS lock status.

`0` The OS Lock is unlocked.

`1` The OS Lock is locked.

RES0, [4:2]

RES0 Reserved.

STICKYPD, [1]

Sticky power down state.

`0` Trace register power has not been removed since the TRCPDSR was last read.

`1` Trace register power has been removed since the TRCPDSR was last read.

This bit is set to 1 when power to the ETM trace unit registers is removed, to indicate that programming state has been lost. It is cleared after a read of the TRCPDSR.

POWER, [0]

Indicates the ETM trace unit is powered:

`0` ETM trace unit is not powered. The trace registers are not accessible and they all return an error response.

`1` ETM trace unit is powered. All registers are accessible.

If a system implementation allows the ETM trace unit to be powered off independently of the debug power domain, the system must handle accesses to the ETM trace unit appropriately.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The TRCPDSR can be accessed through the external debug interface, offset `0x314`.

D10.54 TRCPIDR0, ETM Peripheral Identification Register 0

The TRCPIDR0 provides information to identify a trace component.

Bit field descriptions

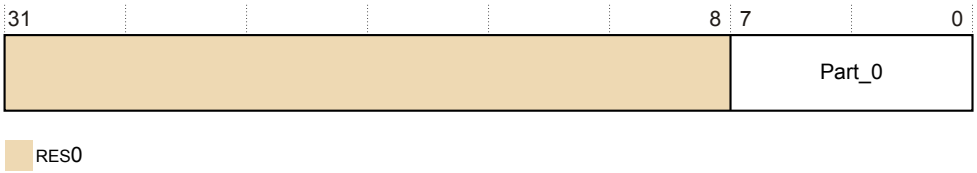

Figure D10-52 TRCPIDR0 bit assignments

RES0, [31:8]

RES0 Reserved.

Part_0, [7:0]

`0x0A` Least significant byte of the ETM trace unit part number.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The TRCPIDR0 can be accessed through the external debug interface, offset `0xFE0`.

D10.55 TRCPIDR1, ETM Peripheral Identification Register 1

The TRCPIDR1 provides information to identify a trace component.

Bit field descriptions

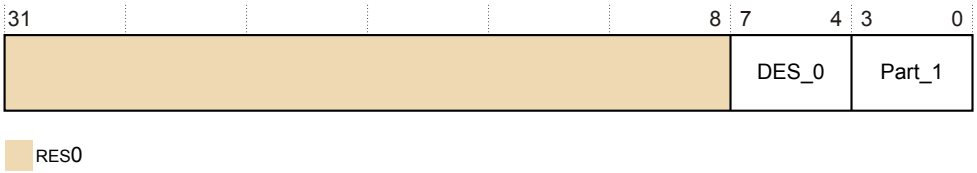

Figure D10-53 TRCPIDR1 bit assignments

RES0, [31:8]

RES0 Reserved.

DES_0, [7:4]

`0xB` Arm Limited. This is bits[3:0] of JEP106 ID code.

Part_1, [3:0]

`0xD` Most significant four bits of the ETM trace unit part number.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The TRCPIDR1 can be accessed through the external debug interface, offset `0xFE4`.

D10.56 TRCPIDR2, ETM Peripheral Identification Register 2

The TRCPIDR2 provides information to identify a trace component.

Bit field descriptions

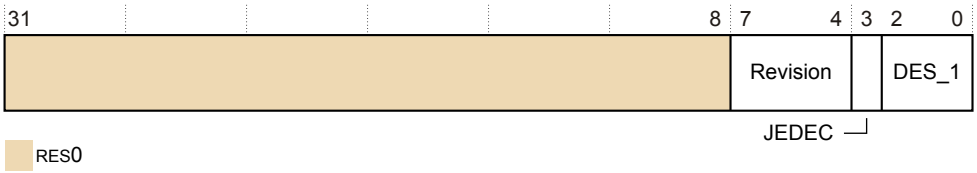

Figure D10-54 TRCPIDR2 bit assignments

RES0, [31:8]

RES0 Reserved.

Revision, [7:4]

`0x2` ETM revision.

JEDEC, [3]

`0b1` RES1. Indicates a JEP106 identity code is used.

DES_1, [2:0]

`0b011` Arm Limited. This is bits[6:4] of JEP106 ID code.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The TRCPIDR2 can be accessed through the external debug interface, offset `0xFE8`.

D10.57 TRCPIDR3, ETM Peripheral Identification Register 3

The TRCPIDR3 provides information to identify a trace component.

Bit field descriptions

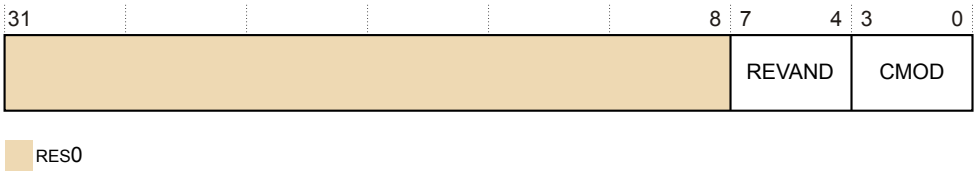

Figure D10-55 TRCPIDR3 bit assignments

RES0, [31:8]

RES0 Reserved.

REVAND, [7:4]

`0x0` Part minor revision.

CMOD, [3:0]

`0x0` Not customer modified.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The TRCPIDR3 can be accessed through the external debug interface, offset `0xFEC`.

D10.58 TRCPIDR4, ETM Peripheral Identification Register 4

The TRCPIDR4 provides information to identify a trace component.

Bit field descriptions

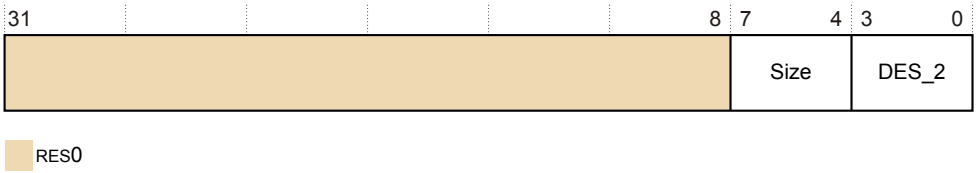

Figure D10-56 TRCPIDR4 bit assignments

RES0, [31:8]

RES0 Reserved.

Size, [7:4]

`0x0` Size of the component. Log2 the number of 4KB pages from the start of the component to the end of the component ID registers.

DES_2, [3:0]

`0x4` Arm Limited. This is bits[3:0] of the JEP106 continuation code.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The TRCPIDR4 can be accessed through the external debug interface, offset `0xFD0`.

D10.59 TRCPIDRn, ETM Peripheral Identification Registers 5-7

No information is held in the Peripheral ID5, Peripheral ID6, and Peripheral ID7 Registers.

They are reserved for future use and are RES0.

D10.60 TRCPRGCTLR, Programming Control Register

The TRCPRGCTLR enables the ETM trace unit.

Bit field descriptions

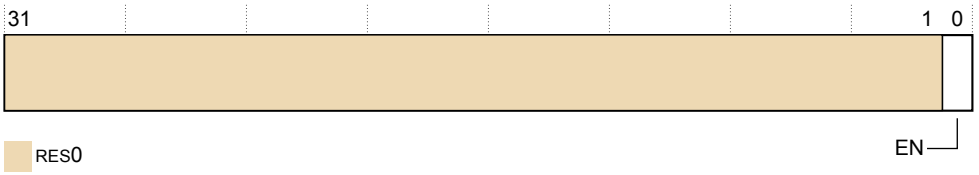

Figure D10-57 TRCPRGCTLR bit assignments

RES0, [31:1]

RES0 Reserved.

EN, [0]

Trace program enable:

`0` The ETM trace unit interface in the core is disabled, and clocks are enabled only when necessary to process APB accesses, or drain any already generated trace. This is the reset value.

`1` The ETM trace unit interface in the core is enabled, and clocks are enabled. Writes to most trace registers are ignored.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The TRCPRGCTLR can be accessed through the external debug interface, offset `0x004`.

D10.61 TRCRSCTLRn, Resource Selection Control Registers 2-16

The TRCRSCTLRn controls the trace resources. There are eight resource pairs, the first pair is predefined as {0,1,pair=0} and having reserved select registers. This leaves seven pairs to be implemented as programmable selectors.

Bit field descriptions

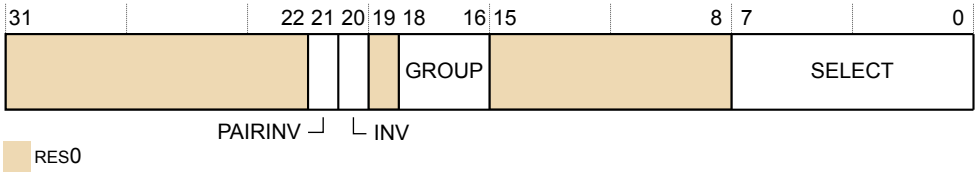

Figure D10-58 TRCRSCTLRn bit assignments

RES0, [31:22]

RES0 Reserved.

PAIRINV, [21]

Inverts the result of a combined pair of resources.

This bit is implemented only on the lower register for a pair of resource selectors.

INV, [20]

Inverts the selected resources:

`0` Resource is not inverted.

`1` Resource is inverted.

RES0, [19]

RES0 Reserved.

GROUP, [18:16]

Selects a group of resources. See the *Arm® ETM Architecture Specification, ETMv4* for more information.

RES0, [15:8]

RES0 Reserved.

SELECT, [7:0]

Selects one or more resources from the required group. One bit is provided for each resource from the group.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The TRCRSCTLRn can be accessed through the external debug interface, offset `0x208-0x023C`.

D10.62 TRCSEQEVRn, Sequencer State Transition Control Registers 0-2

The TRCSEQEVRn defines the sequencer transitions that progress to the next state or backwards to the previous state. The ETM trace unit implements a sequencer state machine with up to four states.

Bit field descriptions

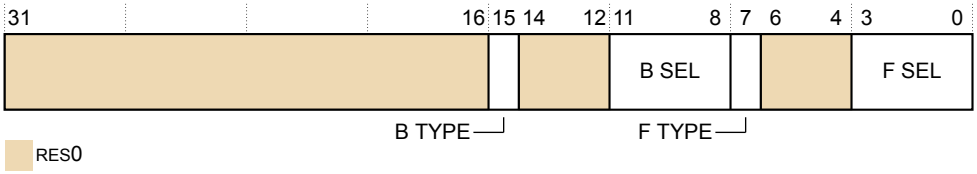

Figure D10-59 TRCSEQEVRn bit assignments

RES0, [31:16]

RES0 Reserved.

B TYPE, [15]

Selects the resource type to move backwards to this state from the next state:

0 Single selected resource.

1 Boolean combined resource pair.

RES0, [14:12]

RES0 Reserved.

B SEL, [11:8]

Selects the resource number, based on the value of B TYPE:

When B TYPE is 0, selects a single selected resource from 0-15 defined by bits[3:0].

When B TYPE is 1, selects a Boolean combined resource pair from 0-7 defined by bits[2:0].

F TYPE, [7]

Selects the resource type to move forwards from this state to the next state:

0 Single selected resource.

1 Boolean combined resource pair.

RES0, [6:4]

RES0 Reserved.

F SEL, [3:0]

Selects the resource number, based on the value of F TYPE:

When F TYPE is 0, selects a single selected resource from 0-15 defined by bits[3:0].

When F TYPE is 1, selects a Boolean combined resource pair from 0-7 defined by bits[2:0].

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The TRCSEQEVRn registers can be accessed through the external debug interface, offsets:

TRCSEQEVR0

0x100.

TRCSEQEVR1

0x104.

TRCSEQEVR2

`0x108`.

D10.63 TRCSEQRSTEVR, Sequencer Reset Control Register

The TRCSEQRSTEVR resets the sequencer to state 0.

Bit field descriptions

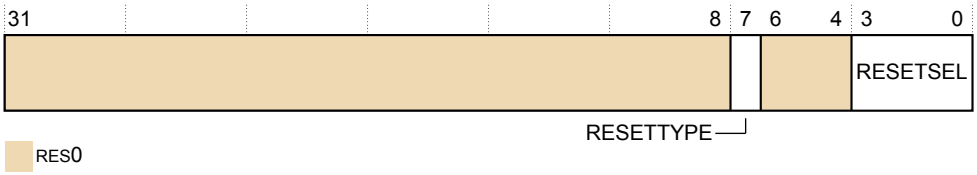

Figure D10-60 TRCSEQRSTEVR bit assignments

RES0, [31:8]

RES0 Reserved.

RESETTYPE, [7]

Selects the resource type to move back to state 0:

`0` Single selected resource.

`1` Boolean combined resource pair.

RES0, [6:4]

RES0 Reserved.

RESETSEL, [3:0]

Selects the resource number, based on the value of RESETTYPE:

When RESETTYPE is 0, selects a single selected resource from 0-15 defined by bits[3:0].

When RESETTYPE is 1, selects a Boolean combined resource pair from 0-7 defined by bits[2:0].

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The TRCSEQRSTEVR can be accessed through the external debug interface, offset `0x118`.

D10.64 TRCSEQSTR, Sequencer State Register

The TRCSEQSTR holds the value of the current state of the sequencer.

Bit field descriptions

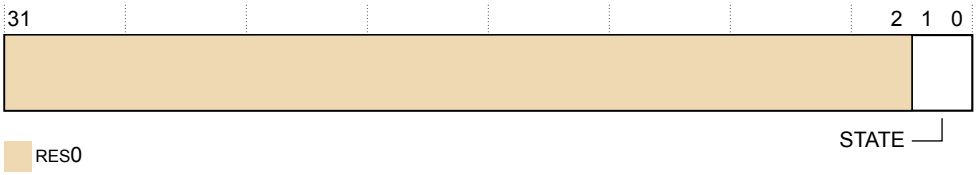

Figure D10-61 TRCSEQSTR bit assignments

RES0, [31:2]

RES0 Reserved.

STATE, [1:0]

Current sequencer state:

`0b00` State 0.

`0b01` State 1.

`0b10` State 2.

`0b11` State 3.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The TRCSEQSTR can be accessed through the external debug interface, offset `0x11c`.

D10.65 TRCSSCCR0, Single-Shot Comparator Control Register 0

The TRCSSCCR0 controls the single-shot comparator.

Bit field descriptions

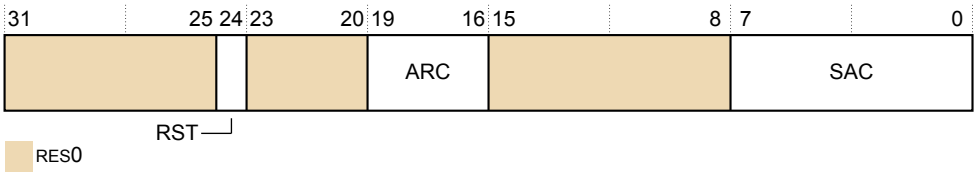

Figure D10-62 TRCSSCCR0 bit assignments

RES0, [31:25]

RES0 Reserved.

RST, [24]

Enables the single-shot comparator resource to be reset when it occurs, to enable another comparator match to be detected:

1 Reset enabled. Multiple matches can occur.

RES0, [23:20]

RES0 Reserved.

ARC, [19:16]

Selects one or more address range comparators for single-shot control.

One bit is provided for each implemented address range comparator.

RES0, [15:8]

RES0 Reserved.

SAC, [7:0]

Selects one or more single address comparators for single-shot control.

One bit is provided for each implemented single address comparator.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The TRCSSCCR0 can be accessed through the external debug interface, offset `0x280`.

D10.66 TRCSSCSR0, Single-Shot Comparator Status Register 0

The TRCSSCSR0 indicates the status of the single-shot comparator. TRCSSCSR0 is sensitive to instruction addresses.

Bit field descriptions

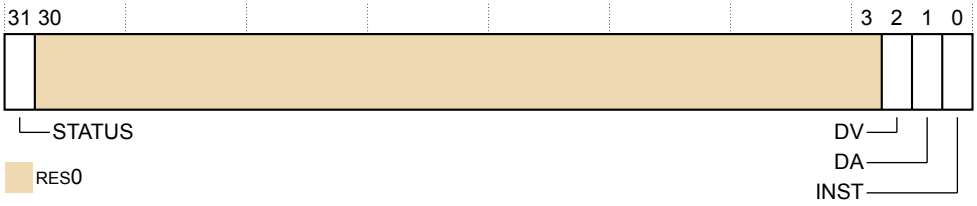

Figure D10-63 TRCSSCSR0 bit assignments

STATUS, [31]

Single-shot status. This indicates whether any of the selected comparators have matched:

`0` Match has not occurred.

`1` Match has occurred at least once.

When programming the ETM trace unit, if TRCSSCCRn.RST is b0, the STATUS bit must be explicitly written to 0 to enable this single-shot comparator control.

RES0, [30:3]

RES0 Reserved.

DV, [2]

Data value comparator support:

`0` Single-shot data value comparisons not supported.

DA, [1]

Data address comparator support:

`0` Single-shot data address comparisons not supported.

INST, [0]

Instruction address comparator support:

`1` Single-shot instruction address comparisons supported.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The TRCSSCSR0 can be accessed through the external debug interface, offset `0x2A0`.

D10.67 TRCSTALLCTLR, Stall Control Register

The TRCSTALLCTLR enables the ETM trace unit to stall the Cortex-A75 core if the ETM trace unit FIFO overflows.

Bit field descriptions

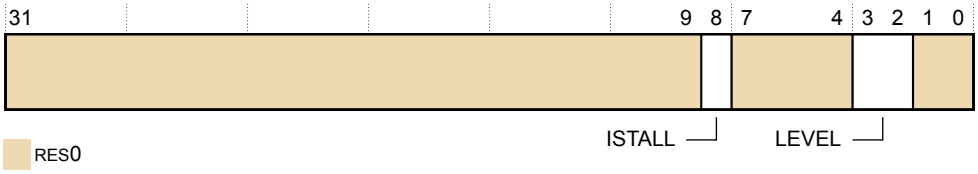

Figure D10-64 TRCSTALLCTLR bit assignments

RES0, [31:9]

RES0 Reserved.

ISTALL, [8]

Instruction stall bit. Controls if the trace unit can stall the core when the instruction trace buffer space is less than LEVEL:

`0` The trace unit does not stall the core.

`1` The trace unit can stall the core.

RES0, [7:4]

RES0 Reserved.

LEVEL, [3:2]

Threshold level field. The field can support 4 monotonic levels from `0b00` to `0b11`, where:

`0b00` Zero invasion. This setting has a greater risk of an ETM trace unit FIFO overflow.

`0b11` Maximum invasion occurs but there is less risk of a FIFO overflow.

RES0, [1:0]

RES0 Reserved.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The TRCSTALLCTLR can be accessed through the external debug interface, offset `0x02c`.

D10.68 TRCSTATR, Status Register

The TRCSTATR indicates the ETM trace unit status.

Bit field descriptions

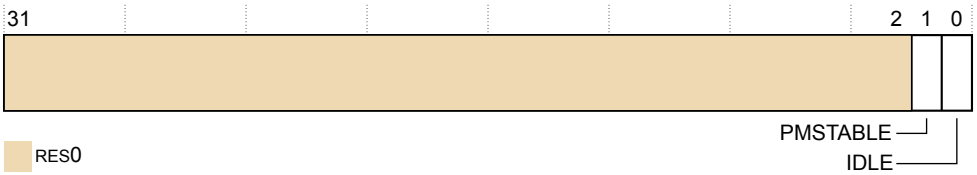

Figure D10-65 TRCSTATR bit assignments

RES0, [31:2]

RES0 Reserved.

PMSTABLE, [1]

Indicates whether the ETM trace unit registers are stable and can be read:

`0` The programmers model is not stable.

`1` The programmers model is stable.

IDLE, [0]

Idle status:

`0` The ETM trace unit is not idle.

`1` The ETM trace unit is idle.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The TRCSTATR can be accessed through the external debug interface, offset `0x00C`.

D10.69 TRCSYNCPR, Synchronization Period Register

The TRCSYNCPR controls how often periodic trace synchronization requests occur.

Bit field descriptions

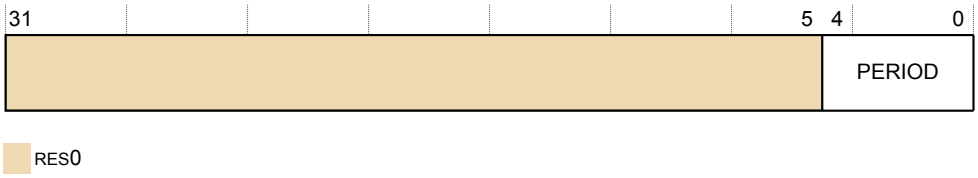

Figure D10-66 TRCSYNCPR bit assignments

RES0, [31:5]

RES0 Reserved.

PERIOD, [4:0]

Defines the number of bytes of trace between synchronization requests as a total of the number of bytes generated by both the instruction and data streams. The number of bytes is 2^N where N is the value of this field:

- A value of zero disables these periodic synchronization requests, but does not disable other synchronization requests.
- The minimum value that can be programmed, other than zero, is 8, providing a minimum synchronization period of 256 bytes.
- The maximum value is 20, providing a maximum synchronization period of 2^{20} bytes.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The TRCSYNCPR can be accessed through the external debug interface, offset `0x034`.

D10.70 TRCTRACEIDR, Trace ID Register

The TRCTRACEIDR sets the trace ID for instruction trace.

Bit field descriptions

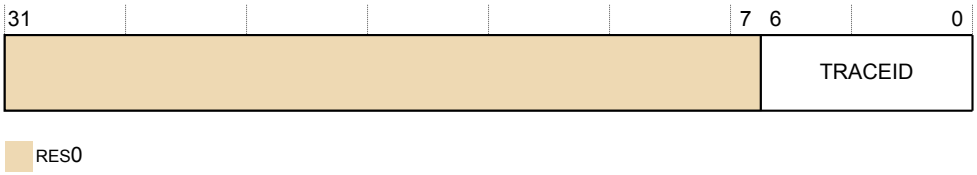

Figure D10-67 TRCTRACEIDR bit Assignments

RES0, [31:7]

RES0 Reserved.

TRACEID, [6:0]

Trace ID value. When only instruction tracing is enabled, this provides the trace ID.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The TRCTRACEIDR can be accessed through the external debug interface, offset `0x040`.

D10.71 TRCTSCTLR, Global Timestamp Control Register

The TRCTSCTLR controls the insertion of global timestamps in the trace streams. When the selected event is triggered, the trace unit inserts a global timestamp into the trace streams. The event is selected from one of the Resource Selectors.

Bit field descriptions

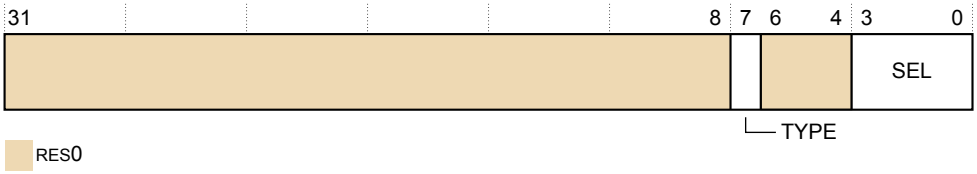

Figure D10-68 TRCTSCTLR bit assignments

RES0, [31:8]

RES0 Reserved.

TYPE, [7]

Single or combined resource selector.

RES0, [6:4]

RES0 Reserved.

SEL, [3:1]

Identifies the resource selector to use.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The TRCTSCTLR can be accessed through the external debug interface, offset `0x030`.

D10.72 TRCVICTLR, ViewInst Main Control Register

The TRCVICTLR controls instruction trace filtering.

Bit field descriptions

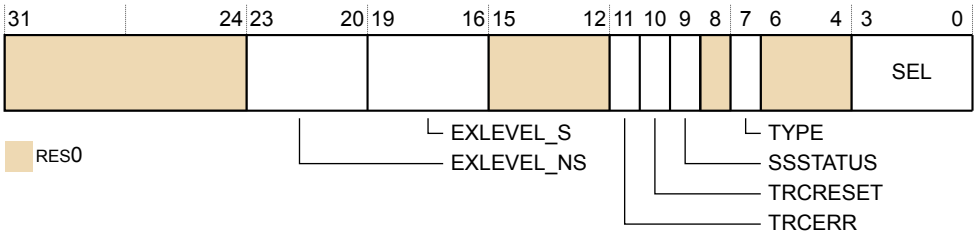

Figure D10-69 TRCVICTLR bit assignments

RES0, [31:24]

RES0 Reserved.

EXLEVEL_NS, [23:20]

In Non-secure state, each bit controls whether instruction tracing is enabled for the corresponding exception level:

0 Trace unit generates instruction trace, in Non-secure state, for exception level *n*.

1 Trace unit does not generate instruction trace, in Non-secure state, for exception level *n*.

The exception levels are:

Bit[20] Exception level 0.

Bit[21] Exception level 1.

Bit[22] Exception level 2.

Bit[23] RAZ/WI. Instruction tracing is not implemented for exception level 3.

EXLEVEL_S, [19:16]

In Secure state, each bit controls whether instruction tracing is enabled for the corresponding exception level:

0 Trace unit generates instruction trace, in Secure state, for exception level *n*.

1 Trace unit does not generate instruction trace, in Secure state, for exception level *n*.

The exception levels are:

Bit[16] Exception level 0.

Bit[17] Exception level 1.

Bit[18] RAZ/WI. Instruction tracing is not implemented for exception level 2.

Bit[19] Exception level 3.

RES0, [15:12]

RES0 Reserved.

TRCERR, [11]

Selects whether a system error exception must always be traced:

0 System error exception is traced only if the instruction or exception immediately before the system error exception is traced.

`1` System error exception is always traced regardless of the value of ViewInst.

TRCRESET, [10]

Selects whether a reset exception must always be traced:

`0` Reset exception is traced only if the instruction or exception immediately before the reset exception is traced.

`1` Reset exception is always traced regardless of the value of ViewInst.

SSSTATUS, [9]

Indicates the current status of the start/stop logic:

`0` Start/stop logic is in the stopped state.

`1` Start/stop logic is in the started state.

RES0, [8]

RES0 Reserved.

TYPE, [7]

Selects the resource type for the viewinst event:

`0` Single selected resource.

`1` Boolean combined resource pair.

RES0, [6:4]

RES0 Reserved.

SEL, [3:0]

Selects the resource number to use for the viewinst event, based on the value of TYPE:

When TYPE is 0, selects a single selected resource from 0-15 defined by bits[3:0].

When TYPE is 1, selects a Boolean combined resource pair from 0-7 defined by bits[2:0].

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The TRCVICTLR can be accessed through the external debug interface, offset `0x080`.

D10.73 TRCVIIECTLR, ViewInst Include-Exclude Control Register

The TRCVIIECTLR defines the address range comparators that control the ViewInst Include/Exclude control.

Bit field descriptions

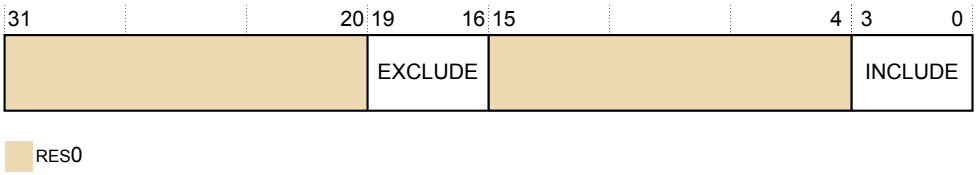

Figure D10-70 TRCVIIECTLR bit assignments

RES0, [31:20]

RES0 Reserved.

EXCLUDE, [19:16]

Defines the address range comparators for ViewInst exclude control. One bit is provided for each implemented Address Range Comparator.

RES0, [15:4]

RES0 Reserved.

INCLUDE, [3:0]

Defines the address range comparators for ViewInst include control.

Selecting no include comparators indicates that all instructions must be included. The exclude control indicates which ranges must be excluded.

One bit is provided for each implemented Address Range Comparator.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The TRCVIIECTLR can be accessed through the external debug interface, offset `0x084`.

D10.74 TRCVISSCTLR, ViewInst Start-Stop Control Register

The TRCVISSCTLR defines the single address comparators that control the ViewInst Start/Stop logic.

Bit field descriptions

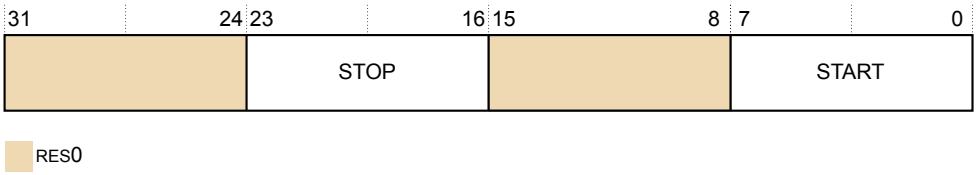

Figure D10-71 TRCVISSCTLR bit assignments

RES0, [31:24]

RES0 Reserved.

STOP, [23:16]

Defines the single address comparators to stop trace with the ViewInst Start/Stop control.

One bit is provided for each implemented single address comparator.

RES0, [15:8]

RES0 Reserved.

START, [7:0]

Defines the single address comparators to start trace with the ViewInst Start/Stop control.

One bit is provided for each implemented single address comparator.

Bit fields and details not provided in this description are architecturally defined. See the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

The TRCVISSCTLR can be accessed through the external debug interface, offset `0x088`.

D10.75 TRCVMIDCVR0, VMID Comparator Value Register 0

The TRCVMIDCVR0 contains a VMID value.

Bit field descriptions

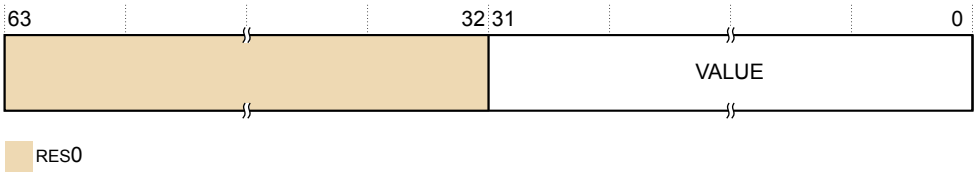

Figure D10-72 TRCVMIDCVR0 bit assignments

RES0, [63:32]

RES0 Reserved.

VALUE, [31:0]

The VMID value.

The TRCVMIDCVR0 can be accessed through the internal memory-mapped interface and the external debug interface, offset `0x640`.

Usage constraints

Accepts writes only when the trace unit is disabled.

Configurations

Available in all configurations.

Attributes

See *D10.1 ETM register summary* on page D10-691.

Part E
Appendices

Appendix A
Cortex®-A75 Core AArch32 unpredictable Behaviors

This appendix describes specific Cortex-A75 core UNPREDICTABLE behaviors of particular interest.

For AArch32 execution, the Armv8-A architecture specifies a much narrower range of legal behaviors for the cases that in Armv7 were described as UNPREDICTABLE. For detailed background information on UNPREDICTABLE behaviors, see the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

It contains the following sections:

A.1 Use of R15 by Instruction

If the use of R15 as a base register for a load or store is UNPREDICTABLE, the value used by the load or store using R15 as a base register is the *Program Counter* (PC) with its usual offset and, in the case of T32 instructions, with the forced word alignment. In this case, if the instruction specifies Writeback, then the load or store is performed without Writeback.

The Cortex-A75 core does not implement a *Read 0* or *Ignore Write* policy on UNPREDICTABLE use of R15 by instruction. Instead, the Cortex-A75 core takes an UNDEFINED exception trap.

A.2 UNPREDICTABLE instructions within an IT Block

Conditional instructions within an IT Block, described as being unpredictable in the *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile* pseudo-code, are executed unconditionally.

The Cortex-A75 core does not implement an unconditional execution policy for the following instructions. Instead all execute conditionally:

- NEON instructions new to Armv8.
- All instructions in the Armv8 Cryptographic Extensions.
- CRC32.

A.3 Load/Store accesses crossing page boundaries

The Cortex-A75 core implements a set of behaviors for load or store accesses that cross page boundaries.

Crossing a page boundary with different memory types or shareability attributes

The *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*, states that a memory access from a load or store instruction that crosses a page boundary to a memory location that has a different memory type or shareability attribute results in CONSTRAINED UNPREDICTABLE behavior.

Crossing a 4KB boundary with a Device access

The *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile*, states that a memory access from a load or store instruction to Device memory that crosses a 4KB boundary results in CONSTRAINED UNPREDICTABLE behavior.

Implementation (for both page boundary specifications)

For an access that crosses a page boundary, the Cortex-A75 core implements the following behaviors:

- Store crossing a page boundary:
  - No alignment fault.
  - The access is split into two stores.
  - Each store uses the memory type and shareability attributes associated with its own address.
- Load crossing a page boundary (Device to Device and Normal to Normal):
  - No alignment fault.
  - The access is split into two loads.
  - Each load uses the memory type and shareability attributes associated with its own address.
- Load crossing a page boundary (Device to Normal and Normal to Device):
  - The instruction might generate an alignment fault.
  - If no fault is generated, the access is split into two loads.
  - Each load uses the memory type and shareability attributes associated with its own address.

A.4 Armv8 Debug UNPREDICTABLE behaviors

This section describes the behavior that the Cortex-A75 core implements when:

- A topic has multiple options.
- The behavior differs from either or both of the Options and Preferences behaviors.

Note

This section does not describe the behavior when a topic only has a single option and the core implements the preferred behavior.

Table A-1 Armv8 Debug UNPREDICTABLE behaviors

Scenario	Behavior
A32 BKPT instruction with condition code not AL	The core implements the following preferred option: • Executed unconditionally.
Address match breakpoint match only on second halfword of an instruction	The core generates a breakpoint on the instruction, unless it is a breakpoint on the second half of the first 32-bit instruction. In this case the breakpoint is taken on the following instruction.
Address matching breakpoint on A32 instruction with DBGBCRn.BAS=1100	The core implements the following option: • Does not match.
Address match breakpoint match on T32 instruction at DBGBCRn+2 with DBGBCRn.BAS=1111	The core implements the following option: • Does match.
Address mismatch breakpoint match on T32 instruction at DBGBCRn +2 with DBGBCRn.BAS=1111	The core implements the following option: • Does not match.
Other mismatch breakpoint matches any address in current mode and state	The core implements the following option: • Immediate Breakpoint debug event.
Mismatch breakpoint on branch to self	The core implements the following option: • Instruction is stepped an UNKNOWN number of times, while it continues to branch to itself.
Link to non-existent breakpoint or breakpoint that is not context-aware	The core implements the following option: • No Breakpoint or Watchpoint debug event is generated, and the LBN field of the *linker* reads UNKNOWN.
DBGWCRn_EL1.MASK!=00000 and DBGWCRn_EL1.BAS!=11111111	The core behaves as indicated in the sole Preference: • DBGWCRn_EL1.BAS is ignored and treated as if `0x11111111`.
Address-matching Vector catch on 32-bit T32 instruction at (vector-2)	The core implements the following option: • Does match.
Address-matching Vector catch on 32-bit T32 instruction at (vector+2)	The core implements the following option: • Does match.
Address-matching Vector catch and breakpoint on same instruction	The core implements the following option: • Report breakpoint.

Table A-1 Armv8 Debug UNPREDICTABLE behaviors (continued)

Scenario	Behavior
Address match breakpoint with DBGBCRn_EL1.BAS=0000	The core implements the following option: • As if disabled.
DBGWCRn_EL1.BAS specifies a non-contiguous set of bytes within a double-word	The core implements the following option: • A Watchpoint debug event is generated for each byte.
A32 HLT instruction with condition code not AL	The core implements the following option: • Executed unconditionally.
Execute instruction at a given EL when the corresponding EDECCR bit is 1 and Halting is allowed	The core behaves as follows: • Generates debug event and Halt no later than the instruction following the next *Context Synchronization operation* (CSO) excluding ISB instruction.
Unlinked Context matching and Address mismatch breakpoints taken to Abort mode	The core implements the following option: • A Prefetch Abort debug exception is generated. Because the breakpoint is configured to generate a breakpoint at PL1, the instruction at the Prefetch Abort vector generates a Vector catch debug event. **Note** The debug event is subject to the same CONSTRAINED UNPREDICTABLE behavior, therefore the Breakpoint debug event is repeatedly generated an UNKNOWN number of times.
Vector catch on Data or Prefetch abort, and taken to Abort mode	The core implements the following option: • A Prefetch Abort debug exception is generated. If Vector catch is enabled on the Prefetch Abort vector, this generates a Vector catch debug event. **Note** The debug event is subject to the same CONSTRAINED UNPREDICTABLE behavior, therefore the Breakpoint debug event is repeatedly generated an UNKNOWN number of times.
H > N or H = 0 at Non-secure EL1 and EL0, including value read from PMCR_EL0.N	The core implements: • A simple implementation where all of HPMN[4:0] are implemented, and In Non-secure EL1 and EL0: — If H > N then M = N. — If H = 0 then M = 0.
H > N or H = 0: value read back in MDCR_EL2.HPMN	The core implements: • A simple implementation where all of HPMN[4:0] are implemented and for reads of MDCR_EL2.HPMN, return H.
P ≥ M and P ≠ 31: reads and writes of PM XEVTYPER_EL0 and PMXEVCNTR_EL0	The core implements: • A simple implementation where all of SEL[4:0] are implemented, and if P ≥ M and P ≠ 31 then the register is RES0.
P ≥ M and P ≠ 31: value read in PMSELR_EL0.SEL	The core implements: • A simple implementation where all of SEL[4:0] are implemented, and if P ≥ M and P ≠ 31 then the register is RES0.

Table A-1 Armv8 Debug UNPREDICTABLE behaviors (continued)

Scenario	Behavior
P = 31: reads and writes of PMXEVCNTR_EL0	The core implements: • RES0.
n ≥ M: Direct access to PMEVCNTRn_EL0 and PMEVTYPERn_EL0	The core implements: • If n ≥ N, then the instruction is UNALLOCATED. • Otherwise if n ≥ M, then the register is RES0.
Exiting Debug state while instruction issued through EDITR is in flight	The core implements the following option: • The instruction completes in Debug state before executing the restart.
Using memory-access mode with a non-word-aligned address	The core behaves as indicated in the sole Preference: • Does unaligned accesses, faulting if these are not permitted for the memory type.
Access to memory-mapped registers mapped to Normal memory	The core behaves as indicated in the sole Preference: • The access is generated, and accesses might be repeated, gathered, split or resized, in accordance with the rules for Normal memory, meaning the effect is UNPREDICTABLE.
Not word-sized accesses or (AArch64 only) doubleword-sized accesses >	The core behaves as indicated in the sole Preference: • Reads occur and return UNKNOWN data. • Writes set the accessed register(s) to UNKNOWN.
External debug write to register that is being reset	The core behaves as indicated in the sole Preference: • Takes reset value.

Table A-1 Armv8 Debug UNPREDICTABLE behaviors (continued)

Scenario	Behavior
Accessing reserved debug registers	The core deviates from preferred behavior because the hardware cost to decode some of these addresses in debug power domain is significantly high. The actual behavior is: 1. For reserved debug registers in the address range `0x000-0xCFC` and Performance Monitors registers in the address range `0x000-0xF00`, the response is either CONSTRAINED UNPREDICTABLE Error or RES0 when any of the following errors occurs: **Off** The core power domain is either completely off or in a low-power state where the core power domain registers cannot be accessed. **DLK** `DoubleLockStatus()` is TRUE and OS double-lock is locked (EDPRSR.DLK is 1). **OSLK** OS lock is locked (OSLSR_EL1.OSLK is 1). 2. For reserved debug registers in the address ranges `0x400-0x4FC` and `0x800-0x8FC`, the response is CONSTRAINED UNPREDICTABLE Error or RES0 when the conditions in *1* do not apply and the following error occurs: **EDAD** `AllowExternalDebugAccess()` is FALSE. External debug access is disabled. 3. For reserved Performance Monitor registers in the address ranges `0x000-0x0FC` and `0x400-0x47C`, the response is either CONSTRAINED UNPREDICTABLE Error, or RES0 when the conditions in *1* and *2* do not apply, and the following error occurs: **EPMAD** `AllowExternalPMUAccess()` is FALSE. External Performance Monitors access is disabled.
Clearing the *clear-after-read* EDPRSR bits when Core power domain is on, and `DoubleLockStatus()` is TRUE	The core behaves as indicated in the sole Preference: • Bits are not cleared to zero.

A.5 Other UNPREDICTABLE behaviors

This section describes other UNPREDICTABLE behaviors.

Table A-2 Other UNPREDICTABLE behaviors

Scenario	Description
CSSELR indicates a cache that is not implemented.	If CSSELR indicates a cache that is not implemented, then on a read of the CCSIDR the behavior is CONSTRAINED UNPREDICTABLE, and can be one of the following: • The CCSIDR read is treated as NOP. • The CCSIDR read is UNDEFINED. • The CCSIDR read returns an UNKNOWN value (preferred).
HDCR.HPMN is set to 0, or to a value larger than PMCR.N.	If HDCR.HPMN is set to 0, or to a value larger than PMCR.N, then the behavior in Non-secure EL0 and EL1 is CONSTRAINED UNPREDICTABLE, and one of the following must happen: • The number of counters accessible is an UNKNOWN non-zero value less than PMCR.N. • There is no access to any counters. For reads of HDCR.HPMN by EL2 or higher, if this field is set to 0 or to a value larger than PMCR.N, the core must return a CONSTRAINED UNPREDICTABLE value that is one of: • PMCR.N. • The value that was written to HDCR.HPMN. • (The value that was written to HDCR.HPMN) modulo 2h, where h is the smallest number of bits required for a value in the range 0 to PMCR.N.

Appendix B
Revisions

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

It contains the following section:

- *B.1 Revisions* on page Appx-B-796.

B.1 Revisions

This section describes the technical changes between released issues of this document.

Table B-1 Issue 0000-00

Change	Location	Affects
First release for r0p0	-	-

Table B-2 Differences between Issue 0000-00 and Issue 0100-00

Change	Location	Affects
First release for r1p0	Document history table.	r1p0
CPUECTLR/CPUECTLR_EL1 bits updated	*B1.18 CPUECTLR, CPU Extended Control Register* on page B1-155 and *B2.26 CPUECTLR_EL1, CPU Extended Control Register, EL1* on page B2-321.	r1p0
Operations in AArch64 registers used to access internal memory table	*A6.6 Direct access to internal memory* on page A6-80.	r1p0

Table B-3 Differences between Issue 0100-00 and Issue 0200-00

Change	Location	Affects
First release for r2p0	Document history table. MIDR reset value in *B1.4 AArch32 registers by functional group* on page B1-131. *B1.74 MIDR, Main ID Register* on page B1-242. MIDR_EL1 reset value in *B2.4 AArch64 registers by functional group* on page B2-286. *B2.83 MIDR_EL1, Main ID Register, EL1* on page B2-408. FPSID reset value in *B5.3 AArch32 register summary* on page B5-530. *B5.4.1 FPSID, Floating-Point System ID Register* on page B5-531. *D3.10 EDPIDR2, External Debug Peripheral Identification Register 2* on page D3-614. *D6.9 PMPIDR2, Performance Monitors Peripheral Identification Register 2* on page D6-655. TRCIDR1 and TRCPIDR2 reset values in *D10.1 ETM register summary* on page D10-691. *D10.30 TRCIDR1, ID Register 1* on page D10-731. *D10.56 TRCPIDR2, ETM Peripheral Identification Register 2* on page D10-760.	r2p0
Activity monitoring feature added. **Note** This feature must be used with the r1p0 revision of the DSU.	*Chapter C3 Activity Monitor Unit* on page C3-565. *Chapter D7 AArch64 AMU registers* on page D7-659. *Chapter D8 Memory-mapped AMU registers* on page D8-671.	r2p0

Table B-3 Differences between Issue 0100-00 and Issue 0200-00 (continued)

Change	Location	Affects
Product name changed from Prometheus to Cortex-A75.	Entire manual.	r2p0
Product name changed from FCM, Flexible Cluster Microarchitecture, to DSU, Dynamic Shared Unit™.	Entire manual.	r2p0
Global terminology change from 'processor' to core for the Cortex-A75 product.	Entire manual.	r2p0
L1 data cache encoding updated.	*A6.6.1 Encoding for tag and data in the L1 data cache* on page A6-80	r2p0
Dot Product instructions introduced in v8.4 added.	*B1.2 AArch32 architectural system register summary* on page B1-123. *B1.4 AArch32 registers by functional group* on page B1-131. *B1.65 ID_ISAR6, Instruction Set Attribute Register 6* on page B1-226. *B2.2 AArch64 architectural system register summary* on page B2-277. *B2.4 AArch64 registers by functional group* on page B2-286. *B2.70 ID_ISAR6_EL1, AArch32 Instruction Set Attribute Register 6, EL1* on page B2-388. *B2.56 ID_AA64ISAR0_EL1, AArch64 Instruction Set Attribute Register 0, EL1* on page B2-363.	r2p0

Table B-4 Differences between Issue 0200-00 and Issue 0201-00

Change	Location	Affects
First release for r2p1	Document history table Product revision in *A1.7 Product revisions* on page A1-34 MIDR reset value in *B1.4 AArch32 registers by functional group* on page B1-131 *B1.74 MIDR, Main ID Register* on page B1-242 MIDR_EL1 reset value in *B2.4 AArch64 registers by functional group* on page B2-286 *B2.83 MIDR_EL1, Main ID Register, EL1* on page B2-408 FPSID reset value in *B5.3 AArch32 register summary* on page B5-530 *B5.4.1 FPSID, Floating-Point System ID Register* on page B5-531 VPIDR reset value in *B1.4 AArch32 registers by functional group* on page B1-131 (same as MIDR)	r2p1
Bit field description updated in PMCEID1_EL0	*D5.3 PMCEID1_EL0, Performance Monitors Common Event Identification Register 1, EL0* on page D5-639.	r2p1
Bit field description updated in DISR_EL1	*B2.35 DISR_EL1, Deferred Interrupt Status Register, EL1* on page B2-338	r2p1
AMU counters information updated	CPUAMCFGR_EL0 reset value in *D7.1 AArch64 AMU register summary* on page D7-660 Bits[7:0] definition in *D7.4 CPUAMCFGR_EL0, Activity Monitors Configuration Register, EL0* on page D7-663	r2p1

Table B-4 Differences between Issue 0200-00 and Issue 0201-00 (continued)

Change	Location	Affects
Introduction to ESB simplified	*A8.5 Error Synchronization Barrier* on page A8-103	r2p1
ARM updated to Arm to reflect company rebranding	Entire manual	r2p1