ARM® Cortex®-A55 Core

Revision: r1p0

Technical Reference Manual

ARM

ARM 100442_0100_00_en

ARM® Cortex®-A55 Core

Technical Reference Manual

Release Information

Document History

Issue	Date	Confidentiality	Change
0000-00	30 September 2016	Confidential	First release for r0p0
0001-00	16 December 2016	Confidential	First release for r0p1
0100-00	23 June 2017	Non-Confidential	First release for r1p0

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®-A55 Core Technical Reference Manual

Part B Register Descriptions

Chapter B1 AArch32 system registers

Part E Appendices

Appendix A AArch32 UNPREDICTABLE Behaviors

Preface

This preface introduces the *ARM® Cortex®-A55 Core Technical Reference Manual*.

It contains the following:

About this book

This Technical Reference Manual is for the Cortex®-A55 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-A55 core.

Chapter A1 Introduction

This chapter provides an overview of the Cortex-A55 core and its features.

Chapter A2 Technical overview

This chapter describes the structure of the Cortex-A55 core.

Chapter A3 Clocks, resets, and input synchronization

This chapter describes the clocks, resets, and input synchronization of the Cortex-A55 core.

Chapter A4 Power management

This chapter describes the power domains and the power modes in the Cortex-A55 core.

Chapter A5 Memory Management Unit

This chapter describes the *Memory Management Unit* (MMU) of the Cortex-A55 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-A55 core.

Chapter A9 Generic Interrupt Controller CPU interface

This chapter describes the Cortex-A55 core implementation of the ARM *Generic Interrupt Controller* (GIC) CPU interface.

Part B Register Descriptions

This part describes the system registers of the Cortex-A55 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.

Part C Debug descriptions
This part describes the debug functionality of the Cortex-A55 core.

Chapter C1 Debug
This chapter describes the debug features of the core.

Chapter C2 PMU
This chapter describes the *Performance Monitor Unit* (PMU).

Chapter C3 ETM
This chapter describes the *Embedded Trace Macrocell* (ETM) for the Cortex-A55 core.

Part D Debug registers
This part describes the debug registers of the Cortex-A55 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 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 D8 ETM registers
This chapter describes the ETM registers.

Part E Appendices
This part describes the appendices of the Cortex-A55 core.

Appendix A AArch32 UNPREDICTABLE Behaviors
This appendix describes the cases in which the Cortex-A55 core implementation diverges from the preferred behavior described in ARMv8 AArch32 UNPREDICTABLE behaviors.

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.

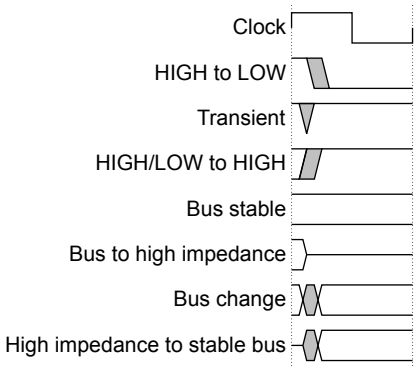


Figure 1 Key to timing diagram conventions

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* (ARM DDI 0487).
- *ARM® Cortex®-A55 Core Advanced SIMD and Floating-point Support Technical Reference Manual* (ARM 100446).
- *ARM® DynamIQ™ Shared Unit Technical Reference Manual* (ARM 100453).
- *ARM® CoreSight™ ELA-500 Embedded Logic Analyzer Technical Reference Manual* (ARM 100127).
- *AMBA® AXI™ and ACE™ Protocol Specification AXI3™, AXI4™, and AXI4-Lite™, ACE and ACE-Lite™* (ARM IHI 0022).
- *AMBA® APB Protocol Version 2.0 Specification* (ARM IHI 0024).
- *ARM® AMBA® CHI Protocol Specification* (ARM IHI 0050).
- *ARM® CoreSight™ Architecture Specification v3.0* (ARM IHI 0029).
- *ARM® Debug Interface Architecture Specification, ADIv5.0 to ADIv5.2* (ARM IHI 0031).
- *AMBA® 4 ATB Protocol Specification* (ARM IHI 0032).
- *ARM® Generic Interrupt Controller Architecture Specification* (ARM IHI 0069).
- *ARM® Embedded Trace Macrocell Architecture Specification ETMv4* (ARM IHI 0064).
- *Low Power Interface Specification ARM® Q-Channel and P-Channel Interfaces* (ARM IHI 0068).

The following confidential documents are only available to licensees:

- *ARM® Cortex®-A55 Core Cryptographic Extension Technical Reference Manual* (ARM 100444).
- *ARM® Cortex®-A55 Core Configuration and Sign-off Guide* (ARM 100443).
- *ARM® Cortex®-A55 Core Integration Manual* (ARM 100445).
- *ARM® DynamIQ™ Shared Unit Integration Manual* (ARM 100403).
- *ARM® DynamIQ™ Shared Unit Configuration and Sign-off Guide* (ARM 100454).

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-A55 Core Technical Reference Manual*.
- The number ARM 100442_0100_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-A55 core and its features.

It contains the following sections:

A1.1 About the core

The Cortex-A55 core is a mid-range, low-power core that implements the ARMv8-A architecture with support for the v8.2 extension, the RAS extension, the Load acquire (LDAPR) instructions introduced in the ARMv8.3 extension, and the Dot Product instructions introduced in the ARMv8.4 extension.

The core has a *Level 1* (L1) memory system, and private *Level 2* (L2) cache. The core is implemented inside the DynamIQ Shared Unit (DSU) as a Little core and is highly configurable with other cores.

The following figure shows an example of a dual-core configuration.

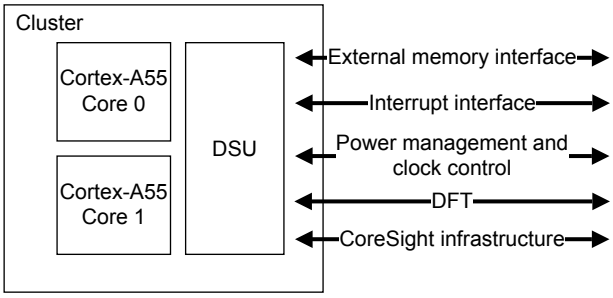


Figure A1-1 Example dual-core configuration with homogeneous cores

The Cortex-A55 core can also be part of a heterogeneous system. The following figure shows an example in which the Cortex-A55 core and another core are integrated into a shared *Level 3* (L3) cluster.

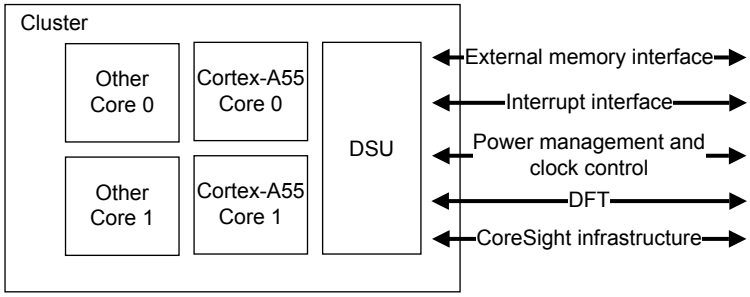


Figure A1-2 Example quad-core configuration with heterogeneous cores

A1.2 Features

The Cortex-A55 core is designed to be the *Little* core in a big.LITTLE™ arrangement and is fully compatible with the big.LITTLE use models introduced by previous generations of Cortex cores.

The Cortex-A55 core includes the following features:

Core Features

- Full implementation of the ARMv8.2-A A64, A32, and T32 instruction sets.
- Both the AArch32 and AArch64 execution states at all Exception levels (EL0 to EL3).
- In-order pipeline with direct and indirect branch prediction.
- Separate L1 data and instruction side memory systems with a *Memory Management Unit* (MMU).
- Support for ARM *TrustZone*® technology.
- Optional Data Engine unit that implements the Advanced SIMD and floating-point architecture support.
- Optional Cryptographic Extension. This architectural extension is only available if the Data Engine is present.
- *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

- Optional unified private L2 cache.
- L1 and L2 cache protection in the form of *Error Correction Code* (ECC) or parity on all RAM instances.

Debug features

- *Reliability, Availability, and Serviceability* (RAS) Extension.
- ARMv8.2 debug logic.
- *Performance Monitoring Unit* (PMU).
- *Embedded Trace Macrocell* (ETM) that supports instruction trace only.

A1.3 Implementation options

The Cortex-A55 core is highly configurable.

Build-time configuration options make it possible to meet functional requirements with the smallest possible area and power. In a configuration with more than one core, all cores have the same build-time configuration except for the L2 cache inclusion and size.

The following table lists the implementation options for a core.

Table A1-1 Implementation options for a core

Feature	Range of options	Notes
L1 instruction cache size	• 16KB • 32KB • 64KB	-
L1 data cache size	• 16KB • 32KB • 64KB	-
L2 cache	• Included • Not included	-
L2 cache size	• 64KB • 128KB • 256KB	-
ECC or parity core cache protection	• Included • Not included	Not available if the L3 cache is implemented without L3 cache protection.
Advanced SIMD and floating-point support	• Included • Not included	There is no option to implement floating-point without Advanced SIMD.
Cryptographic Extension	• Included • Not included	There is no option to implement the Cryptographic Extension without the Advanced SIMD and floating-point support.
CoreSight Embedded Logic Analyzer (ELA)	• Optional support	Support for integrating CoreSight ELA-500. The CoreSight ELA-500 is a separately licensable product.
CoreSight ELA RAM address size	2-25	See the *ARM® CoreSight™ ELA-500 Embedded Logic Analyzer Technical Reference Manual* for more details about the RAM sizing.
Dot Product instruction support	• Included • Not included	Support for the ARMv8.4 SDOT and UDOT instructions. There is no option to implement the Dot Product instructions without the Advanced SIMD and floating point support.

A1.4 Supported standards and specifications

The Cortex-A55 core implements the ARMv8-A architecture and some architecture extensions. It also supports various 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. • Advanced SIMD and floating-point support. • Cryptographic Extension. • RAS Extension. • v8.3 LDPAR instructions. • v8.4 Dot Product instructions.	• You cannot implement floating-point without Advanced SIMD. • You cannot implement the Cryptographic Extension without the Advanced SIMD and floating-point support. • The Cortex-A55 core implements the `LDPAR` instructions introduced in the v8.3 extensions. • The Cortex-A55 core optionally implements the `SDOT` and `UDOT` instructions introduced in the v8.4 extensions.
Generic Interrupt Controller	GICv4	-
PMU	PMUv3	-
Debug	ARMv8-A	With support for the debug features added by the v8.2 extensions.
CoreSight	CoreSightv3	-
Embedded Trace Macrocell	ETMv4.2	-

A1.5 Test features

The Cortex-A55 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-A55 core is delivered as a synthesizable *Register Transfer Level* (RTL) description in Verilog HDL. Before you can use the Cortex-A55 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.

r0p1 Further development and optimization of the product.

r1p0 Addition of the Dot Product instructions introduced in the v8.4 architecture extensions.

Chapter A2
Technical overview

This chapter describes the structure of the Cortex-A55 core.

It contains the following sections:

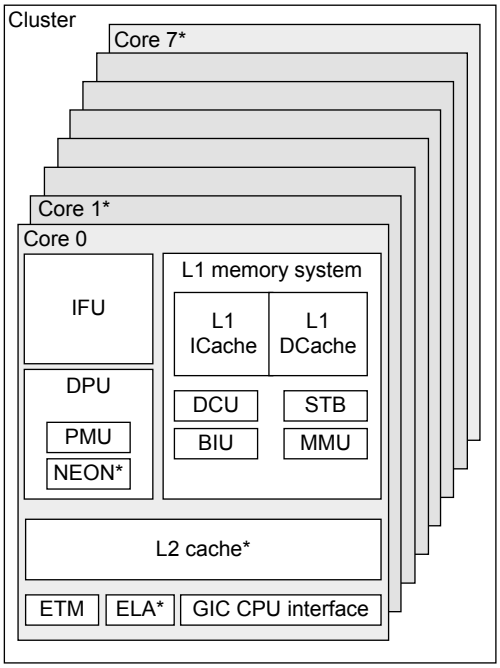
A2.1 Components

The cluster consists of:

- One to eight cores.
- The DynamIQ Shared Unit (DSU), which connects the cores to an external memory system.

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

The following figure includes a top-level functional diagram of a core.



*Optional

Figure A2-1 Core block diagram

Instruction Fetch Unit (IFU)

The IFU fetches instructions from the instruction cache or from external memory and predicts the outcome of branches in the instruction stream. It passes the instructions to the *Data Processing Unit* (DPU) for processing.

Data Processing Unit (DPU)

The DPU decodes and executes instructions. It executes instructions that require data transfer to or from the memory system by interfacing to the *Data Cache Unit* (DCU). The DPU includes the PMU, the Advanced SIMD and floating-point support, and the Cryptographic Extension.

PMU

The PMU provides six performance monitors that can be configured to gather statistics on the operation of each core and the memory system. The information can be used for debug and code profiling.

Advanced SIMD and floating-point support

Advanced SIMD is a media and signal processing architecture that adds instructions primarily for audio, video, 3D graphics, image and speech processing. The floating-point architecture provides support for single-precision and double-precision floating-point operations.

All scalar floating-point instructions are available in the A64 instruction set.

All VFP instructions are available in the A32 and T32 instruction sets.

The A64 instruction set offers additional Advanced SIMD instructions, including double-precision floating-point vector operations.

——— **Note** ———

The Advanced SIMD architecture, its associated implementations, and supporting software, are also referred to as NEON™ technology.

———————————

Cryptographic Extension

The optional Cortex-A55 core Cryptographic Extension supports the ARMv8 Cryptographic Extensions. It is a configuration option that can be set when configuring and integrating the core into a system and applies to all cores. The Cryptographic Extension adds new instructions to Advanced SIMD that accelerate:

- *Advanced Encryption Standard* (AES) encryption and decryption.
- The *Secure Hash Algorithm* (SHA) functions SHA-1, SHA-224, and SHA-256.
- Finite field arithmetic used in algorithms such as *Galois/Counter Mode* and *Elliptic Curve Cryptography*.

Memory Management Unit (MMU)

The MMU provides fine-grained memory system control through a set of virtual-to-physical address mappings and memory attributes that are held in translation tables. These are saved into the *Translation Lookaside Buffer* (TLB) when an address is translated. The TLB entries include global and *Address Space Identifiers* (ASIDs) to prevent context switch TLB flushes. They also include *Virtual Machine Identifiers* (VMIDs) to prevent TLB flushes on virtual machine switches by the hypervisor.

L1 TLBs

The first level of caching for the translation table information is an L1 TLB. It is implemented on each of the instruction and data sides. All TLB-related maintenance operations result in flushing both the instruction and data L1 TLBs.

L2 TLB

A unified L2 TLB handles the misses from the L1 TLBs.

In implementations with core cache protection, parity bits protect the TLB RAMs by enabling the detection of any single-bit error. If an error is detected, the entry is invalidated and fetched again.

L1 memory system

The L1 memory system includes the DCU, the *Store Buffer* (STB), and the *Bus Interface Unit* (BIU).

DCU

The DCU manages all load and store operations.

The L1 data cache RAMs are protected using *Error Correction Codes* (ECC). The ECC scheme is *Single Error Correct Double Error Detect* (SECDED). The DCU includes a combined local and global exclusive monitor that is used by Load-Exclusive and Store-Exclusive instructions.

STB

The STB holds store operations when they have left the load/store pipeline in the DCU and have been committed by the DPU. The STB can request access to the L1 data cache, initiate linefills, or write to L2 and L3 memory systems.

The STB is also used to queue maintenance operations before they are broadcast to other cores in the cluster.

BIU

The BIU contains the interface to the L2 memory system and buffers to decouple the interface from the L1 data cache and STB.

L2 Memory System

The L2 memory system contains the L2 cache. The L2 cache is optional and private to each core. The L2 cache is 4-way set associative, supports 64-byte cache lines, and has a configurable cache RAM size between 64KB and 256KB. The L2 memory system is connected to the DynamIQ Shared Unit through an optional asynchronous bridge.

GIC 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.

DynamIQ™ Shared Unit

The DynamIQ Shared Unit contains the L3 cache and logic required to maintain coherence between the cores in the cluster. For more information, see the *ARM® DynamIQ™ Shared Unit Technical Reference Manual*

Debug and trace components

The Cortex-A55 core supports a range of debug, test, and trace options including:

- Six performance event counters, provided by the PMU, and one cycle counter.
- Six hardware breakpoints, and four watchpoints.
- Per-core instruction trace only ETM.
- Per-core support for an ELA-500.
- *AMBA 4* APB interfaces between the cluster and the Debug Block.

Details of the core-specific debug elements can be found in this document. For information on the cluster debug and trace components supported by the Cortex-A55 core, see the *ARM® DynamIQ™ Shared Unit Technical Reference Manual*.

Related references

Chapter A5 Memory Management Unit on page A5-61.
Chapter A6 Level 1 memory system on page A6-71.
Chapter A7 Level 2 memory system on page A7-89.
Chapter A9 Generic Interrupt Controller CPU interface on page A9-105.
Chapter C1 Debug on page C1-519.
Chapter C2 PMU on page C2-527.
Chapter C3 ETM on page C3-549.

A2.2 Interfaces

The Cortex-A55 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-A55 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-A55 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-A55 core.

It contains the following sections:

A3.1 About Clocks, Resets, and Input Synchronization

The Cortex-A55 core supports hierarchical clock gating.

The Cortex-A55 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-A55 core.

It contains the following sections:

A4.1 About power management

The Cortex-A55 core provides mechanisms to control both dynamic and static power dissipation.

The dynamic power management includes the following features:

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

The static power management includes the following features:

- Dynamic retention.
- Powerdown.

Related references

A4.3 Power domains on page A4-48.
A4.5 Power control on page A4-52.

A4.2 Voltage domains

The Cortex-A55 core supports a VCPU voltage domain and a VSYS voltage domain.

The following figure shows the VCPU and VSYS voltage domains in each Cortex-A55 core and in the DSU. The example shows a configuration with four Cortex-A55 cores.

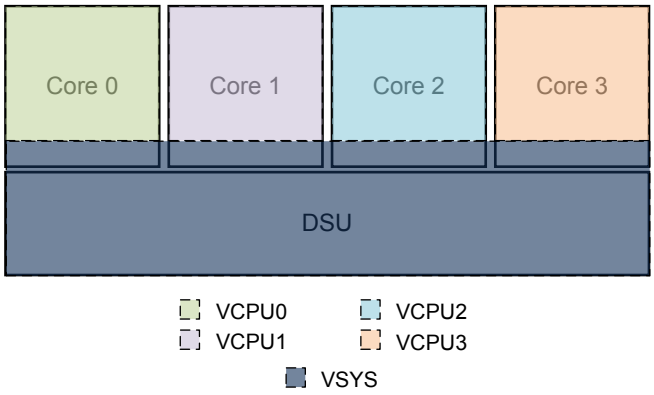


Figure A4-1 Cortex-A55 Voltage Domains

Asynchronous bridge logic exists between the voltage domains. The Cortex-A55 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 Cortex-A55 core supports multiple power domains.

The following figure shows the power domains in the Cortex-A55 core. The colored boxes indicate the PDADVSIMD, PDCPU, and PDSYS power domains, with respective voltage domains shown in dotted lines.

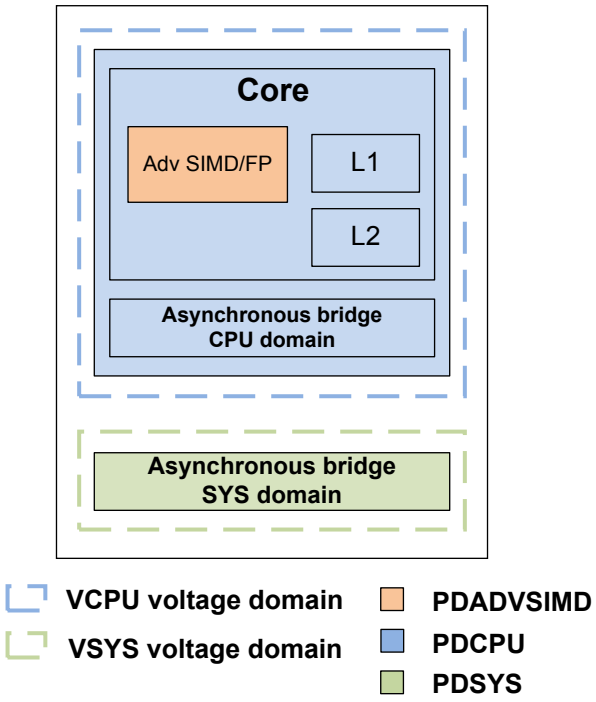


Figure A4-2 Cortex-A55 Core Power Domains

The following figure shows the power domains in the Cortex-A55 cluster, where everything in the same color is part of the same power domain. The example shows four cores in the cluster. The number of cores can vary and the number of domains increase based on the number of cores present. This example only shows the power domains that are associated with the Cortex-A55 cores, and not the other power domains required for a cluster.

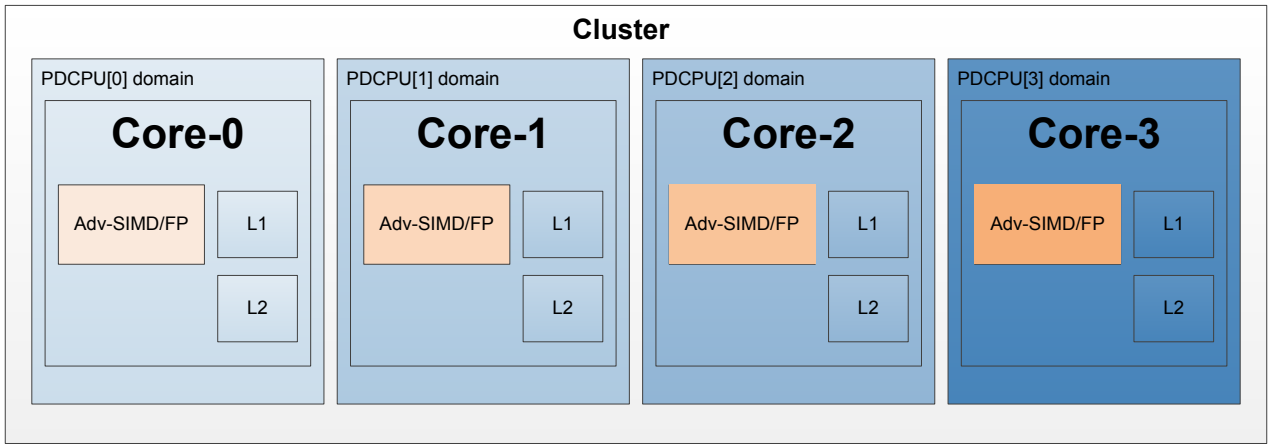


Figure A4-3 Cortex-A55 Power Domains

Note

You do not need to use the full flexibility that the Cortex-A55 clock, voltage, and power domains provide.

The Advanced SIMD and floating-point block in each core is also part of the power domain for that core. However, to support independent retention control, each Advanced SIMD and floating-point block also has its own power domain for isolation from the surrounding domain.

The following table shows the power domains that the Cortex-A55 core supports.

Table A4-1 Power domain description

Power domain	Description
PDCPU<n>	This domain contains all ananke_cpu logic and cpu clock domain logic of the asynchronous bridge. It also includes the optional Advanced SIMD and floating-point block, the L1 and L2 TLBs, L1 and L2 core RAMs, and debug registers that are associated with the core. <n> where n is the core number in the range 0-7. The number represents core 0, core 1, core 2, to core 7. If a core is not present, the corresponding power domain is not present.
PDADVSIMD<n>	This is an optional power domain for Advanced SIMD and floating-point block to implement dynamic retention. <n> where n is the core number in the range 0-7. The number represents core 0, core 1, core 2, to core 7. If a core is not present, the corresponding power domain is not present.
PDSYS	This domain contains the cluster clock domain logic of the asynchronous bridge.

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

A4.4 Architectural clock gating modes

When the Cortex-A55 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*.

Related references

Helios power down sequence.

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-A55 core.

There is one P-Channel per core, plus one P-Channel for the cluster. The Cortex-A55 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-A55 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 Power modes

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

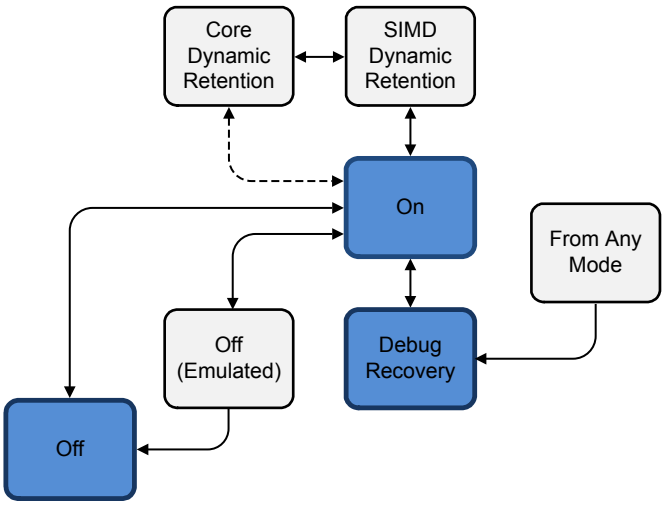


Figure A4-4 Cortex-A55 Core Power Domain Mode Transitions

The darker (blue) blocks indicate the modes that the core can be initialized into. The dotted line transition from On to Core Dynamic Retention is only allowed if SIMD retention is not implemented or has been disabled.

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 table describes the power modes, and the corresponding supported power domain states for individual cores.

Caution

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

Table A4-2 Supported core power mode and power domain states

Power mode	Power domain		Description
	PDCPU	PDADVSIMD	
Debug recovery	On	On	Core on. Advanced SIMD and floating-point block on. Block is active.
On	On	On	Core on. Advanced SIMD and floating-point block on. Block is active.
SIMD dynamic retention	On	Ret	Core on. Advanced SIMD and floating-point block in retention. Block is active.
Core dynamic retention	Ret	Ret	Core retention. Core logic and Advanced SIMD and floating-point block in retention. Logic and RAM retention power only.
Off (emulated)	On	On	Core on. Advanced SIMD and floating-point block on. Block is active.
Off	Off	Off	Core off. Power to the block is gated.

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.

This section contains the following subsections:

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-A55 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 SIMD dynamic retention

In this mode, the Advanced SIMD and floating-point logic is in retention (inoperable but with state retained) and the remainder of the core logic is operational.

This means that if an Advanced SIMD and floating-point instruction is executed while in this mode, it is stalled until the core enters the On mode.

When the Advanced SIMD and floating-point logic is in retention, the clock to the logic is automatically gated outside of the retained domain.

The SIMD dynamic retention is controlled by the CPUPWRCTLR.SIMD_RET_CTRL register.

Related references

B2.30 CPUPWRCTLR_EL1, Power Control Register, EL1 on page B2-326.

A4.6.5 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.

Note

If SIMD dynamic retention is implemented and enabled, then the core does not indicate on **PACTIVE** that it can enter core dynamic retention until it is already in SIMD dynamic retention.

Related references

B2.30 CPUPWRCTLR_EL1, Power Control Register, EL1 on page B2-326.

A4.6.6 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 A4-3 Core Power Modes 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.
SIMD Dynamic Retention	FUNC_RET	7	`0b000111`	SIMD logic is in retention and inoperable. All other logic is on and operational.
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 down sequence

The Cortex-A55 core uses the following power down sequence.

To power down a core, perform the following programming sequence:

1. Save all architectural state.
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 *Instruction Synchronization Barrier* (ISB) instruction.
5. Execute a WFI instruction.

After executing WFI and then receiving a powerdown request from the power controller, the hardware performs the following:

- Disabling and flushing of caches (L1 and L2).
- Removal of the core from coherency.

Note

When the CPUPWRCTLR.CORE_PWRDN_EN bit is set, executing a WFI instruction automatically masks all interrupts and wake-up events in the core. As a result, applying reset is the only way to wake up the core from this WFI.

Related references

B2.30 CPUPWRCTLR_EL1, Power Control Register, EL1 on page B2-326.

A4.9 Debug over powerdown

The Cortex-A55 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-A55 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 page 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 L1 TLB	15 entries, fully associative
Data L1 TLB	16 entries, fully associative
L2 TLB	1024 entries, 4-way set associative
Walk cache RAM	64 entries, 4-way set associative
IPA cache RAM	64 entries, 4-way set associative

L2 TLB entries contain global and *Address Space Identifiers* (ASID) to prevent context switch TLB flushes.

The TLB entries contain a *Virtual Machine Identifier* (VMID) to prevent context switch TLB flushes on virtual machine switches by the hypervisor.

The Cortex-A55 core supports a 40-bit physical address range, which allows 1TB of physical memory to be addressed.

A5.1.2 AAarch32 and AArch64 behavior differences

The Cortex-A55 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 AAarch32 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.
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	40 bits only.	Maximum 40 bits. Any configuration of TCR_ELx.IPS over 40 bits is considered as 40 bits. You can enable or disable each stage of the address translation independently.

The Cortex-A55 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 *Translation Lookaside Buffer* (TLB) is a cache of recently executed page translations within the MMU. The Cortex-A55 core implements a two-level TLB structure. The L2 TLB stores all page sizes and is responsible for breaking these down into smaller pages when required for the data-side or instruction-side L1 TLB.

TLB lockdown is not supported.

After reset, an Invalidate All operation is executed and all entries in the TLB are invalidated.

A5.2.1 L1 TLB

The first level of caching for the translation table information is an L1 TLB, implemented on each of the instruction and data sides.

The Cortex-A55 L1 instruction TLB supports 4KB, 16KB, 64KB, and 2MB pages.

The Cortex-A55 L1 data TLB supports 4KB pages only.

Any other page sizes are fractured after the L2 TLB and the appropriate page size sent to the L1 TLB.

All TLB maintenance operations affect both the L1 instruction and data TLBs and cause them to be invalidated.

A5.2.2 L2 TLB

A unified L2 TLB handles any misses from the L1 instruction and data TLBs.

- A 4-way, set-associative, 1024 entry cache.
- Supports all *Virtual Memory System Architecture* (VMSA) v8 block sizes, except for 1GB.

If a 1GB block is fetched, it is split into 512MB blocks and the appropriate block for the lookup is stored.

Accesses to the L2 TLB take a variable number of cycles, based on:

- Competing requests from the L1 TLBs.
- TLB maintenance operations in flight.
- Different page size mappings in use.

A5.2.3 IPA cache RAM

The IPA cache RAM holds mappings between *intermediate physical addresses* (IPAs) and *physical addresses* (PAs).

Only Non-secure EL1 and EL0 stage 2 translations use the IPA cache. When a stage 2 translation completes, the cache is updated. The IPA cache is checked whenever a stage 2 translation is required.

Like the L2 TLB, the IPA cache RAM can hold entries for different sizes.

A5.2.4 Walk cache RAM

The walk cache RAM holds the result of a stage 1 translation up to, but not including, the last level.

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 that is 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 *Address Space Identifier*ASID or is global. In addition, each TLB entry contains a field to store the *Virtual Machine Identifier* (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 only.
- EL2 (or EL0 in VHE mode) in Non-secure state.
- EL1 or EL0 in Secure state or EL3 in Secure state in AArch32.
- EL1 or EL0 in Non-secure state.

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

- When VA[48:N], matches the requested address, where N is $\log_2$ of the block size for that translation that is stored in the TLB entry, moderated by the page size.
- When the memory space matches the memory space state of the requests. The memory space can be one of the four states mentioned above.
- The ASID matches the current ASID held in the CONTEXTIDR, TTBR0, or TTBR1 register, or the entry is marked global.
- The ASID matches are ignored for requests originating from EL2 when not in VHE mode or from EL3 in AArch64.
- The VMID matches the current VMID held in the VTTBR_EL2 register.
- The VMID match is ignored for a request not originating from Non-secure EL0 or EL1.

A5.4 Translation table walks

When the Cortex-A55 core generates a memory access, the MMU:

1. Performs a lookup for the requested VA and current translation regime in the relevant instruction or data L1 TLB.
2. If there is a miss in the relevant L1 TLB, the MMU performs a lookup for the requested VA, current ASID, current VMID, and translation regime in the L2 TLB.
3. If there is a miss in the L2 TLB, the MMU performs a hardware translation table walk.

In the case of an L2 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

Translation table walk can be performed in cacheable or non-cacheable regions. This is determined by the translation table walk memory attribute, which can be affected by several different configurations:

- IRGN and ORGN bits in the TCR_ELx and VTCR_EL2 registers (or TTBR0/TTBR1_ELx register for short-descriptor translation table format), which define the memory type for translation table walk.
- SCTRL_ELx.C and HCR_EL2.CD or HCR.CD, which affect the table walk to cacheable or non-cacheable memory.
- Stage 2 memory attribute for stage 1 translation table walk, which affect the stage 1 translation table walk memory attribute.

For more information on the control fields, see the *ARM® Architecture Reference Manual ARMv8, for ARMv8-A architecture profile*.

Only when the final translation table walk memory attribute is inner write-back and outer write-back and the cache is enabled, the translation table walk accesses the cacheable memory.

A5.5.2 Hardware management of the Access flag and dirty state

The Cortex-A55 core includes the option to perform hardware updates to the translation tables in AArch64 state only.

These features are enabled in registers TCR_ELx and VTCR_EL2. 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 updates to the Access flag and to dirty state only when the translation tables are held in Inner Write-Back, Outer Write-Back Normal memory regions.

If software requests a hardware update in a region that is not Inner Write-Back or Outer Write-Back Normal memory, 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.

For the Cortex-A55 core, the following situations can cause hardware updates to the Access flag or to dirty state:

- For a Store-Exclusive instruction to a memory location for which the DBM bit is 1 and the stage 1 AP[2] bit is 1, if the Store-Exclusive fails because the exclusive monitor is not in the exclusive state, the AP[2] bit in the translation table is updated.
- For a Store-Exclusive instruction to a memory location for which the DBM bit is 1, and the stage 2 S2AP[1] bit is 0, if the Store-Exclusive fails because the exclusive monitor is not in the exclusive state, the S2AP[1] bit in the translation table is updated.
- For a store to a memory location for which the DBM bit is 1, and the stage 1 AP[2] bit is 1, the AP[2] bit in the translation table is updated:
  - If the memory location generates a synchronous external abort on a write for a store to a memory location.
  - If the memory location generates a watchpoint on a write.
- For a store to a memory location for which the DBM bit is 1, and the stage 2 S2AP[1] bit is 0, the S2AP[1] bit in the translation table is updated:
  - If the memory location generates a synchronous external abort on a write for a store to a memory location.
  - If the memory location generates a watchpoint on a write.

- For a CAS or CASP instruction to a memory location for which the DBM bit is 1, and the stage 1 AP[2] bit is 1, if the compare fails, and the location is not updated, the AP[2] bit in the translation table is updated.
- For a CAS or CASP instruction to a memory location for which the DBM bit is 1, and the stage 2 S2AP[1] bit is 0, if the compare fails, and the location is not updated, the S2AP[1] bit in the translation table is updated.

For more information about hardware updates of the Access flag and dirty state, see the *ARM® Architecture Reference Manual ARMv8, for ARMv8-A architecture profile*.

A5.6 Responses

Certain faults and aborts can cause an exception to be taken because of a memory access.

A5.6.1 MMU responses

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

- An L1 TLB hit.
- An L2 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.

A5.6.2 MMU aborts

The MMU can detect faults that are related to address translation and can cause exceptions to be taken to the processing element. Faults can include address size, translation, access flags, and permissions.

See the *ARM® Architecture Reference Manual ARMv8, for ARMv8-A architecture profile* for more information about aborts.

A5.6.3 External aborts

External aborts are aborts that occur in the memory system rather than aborts that the MMU detects. Normally, external memory aborts are rare. External aborts are caused by errors flagged by the external memory interfaces or are generated because of an uncorrected ECC error in the L1 data cache or L2 cache arrays.

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 a Store Multiple operation, the address captured in the fault register is that of the address that generated the synchronous external abort.

A5.6.4 Mis-programming contiguous hints

A programmer might mis-program the translation tables so that the block size being used to translate the address is larger than the size of the input address. Or a programmer might mis-program the translation tables so that the address range translated by a set of blocks marked as contiguous, by use of the contiguous bit, is larger than the size of the input address.

If there is this kind of mis-programming the Cortex-A55 core will not generate a translation fault.

A5.6.5 Conflict aborts

Conflict aborts are generated from the L1 TLB. If a conflict abort is detected in the L2 TLB, it will choose one valid translation it will not generate a conflict abort.

See also the *ARM® Architecture Reference Manual ARMv8, for ARMv8-A architecture profile* for more information.

A5.6.6 Memory Behavior

The Cortex-A55 core supports all the ARMv8 memory types.

However, the following behaviors are simplified and so for best performance their use is not recommended:

Write-Through Memory that is marked as Write-Through cannot be cached on the data-side and does not make coherency requests. On the instruction-side, areas that are marked as Write-Through and Write-Back can be cached in the L1 instruction cache. However, only areas marked as Write-Back can be cached in the L2 cache or the L3 cache.

Mixed inner and outer cacheability Memory that is not marked as inner and outer Write-Back cannot be cached on the data-side and does not make coherency requests. This applies to the memory type only, and not to the allocation hints. All caches within the cluster are treated as being part of the inner cacheability domain.

For more information on supported memory behaviors, see the *ARM® Architecture Reference Manual ARMv8, for ARMv8-A architecture profile*

A5.6.7 Support for ARMv8 device memory types

The ARMv8 architecture includes memory types that replace the ARMv7 Device and Strongly-ordered memory types. These device memory types have the following three attributes:

G – Gathering
The capability to gather and merge requests together into a single transaction.

R – Reordering
The capability to reorder transactions.

E – Early Write Acknowledgement
The capability to accept early acknowledge of transactions from the interconnect.

The legal combinations are described in the following table:

Table A5-3 ARMv8 Device Memory Types

Memory Type	Cortex-A55 Support	Comment
GRE	Yes	Similar to ‘Normal’ non-cacheable, but does not permit speculative accesses.
nGRE	Yes	Transactions may be re-ordered within the L3 memory system, or in the system interconnect.
nGnRE	Yes	Corresponds to ‘Device’ in ARMv7
nGnRnE	Yes	Corresponds to ‘Strongly Ordered’ in ARMv7 Treated the same as nGnRE inside Cortex-A55, but reported differently on the bus interface.

For more information, see the *ARM® Architecture Reference Manual ARMv8, for ARMv8-A architecture profile*.

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-A55 core's L1 memory system enhances core performance and power efficiency.

It consists of separate instruction and data caches. You can configure instruction and data caches independently during implementation to sizes of 16KB, 32KB, or 64KB.

L1 instruction-side memory system

The L1 instruction-side memory system provides an instruction stream to the DPU. Its key features are:

- 64-byte instruction side cache line length.
- 4-way set associative L1 instruction cache.
- 128-bit read interface to the L2 memory system.

The Cortex-A55 core uses extensive branch prediction to improve *Instructions Per Clock* (IPC) and power efficiency.

L1 data-side memory system

The L1 data-side memory system responds to load and store requests from the DPU. It also responds to SCU snoop requests from other cores, or external masters. Its key features are:

- 64-byte data side cache line length.
- 4-way set associative L1 data cache.
- Read buffer that services both the *Data Cache Unit* (DCU), and the *Instruction Fetch Unit* (IFU).
- 64-bit read path from the data L1 memory system to the datapath.
- 128-bit write path from the datapath to the L1 memory system.
- Merging store buffer capability which writes to all types of memory (device, normal cacheable and normal non-cacheable).
- Data side prefetch engine that detects patterns of strides with multiple streams are allowed in parallel, capable of detecting both constant and patterns of strides.

A6.2 Cache behavior

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

- On a cache miss, the cache performs a critical word-first fill.

A6.2.1 Instruction cache disabled behavior

If the instruction cache is disabled, all instruction fetches to cacheable memory are treated as if they were non-cacheable.

This means that instruction fetches might not be coherent with caches in other cores, and software must take account of this.

- In AArch64 state, lines may still be allocated into the instruction cache even if the memory is marked non-cacheable or the instruction cache is disabled. See the *ARM® Architecture Reference Manual ARMv8, for ARMv8-A architecture profile* for more information.
- In AArch32 state, lines are not allocated into instruction cache when the instruction cache is disabled. Allocation into the instruction cache only occurs when the instruction cache is enabled, and memory is marked as Write-Back or Write-Through cacheable.

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 SCTLR.C bit is set to 0, load and store instructions do not access any of the L1 data, L2, or DSU L3 caches.

The SCTLR.C bit controls whether accesses from the core can look up and allocate into the data cache and unified L2 or L3 caches. Data cache maintenance operations execute normally, regardless of how the SCTLR.C bit is set.

If the SCTLR.C bit is set to 0, then the following apply:

- Instruction fetches cannot allocate in the L2 or L3 caches.
- All load 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 and DC IVAC instructions in AArch64 perform an invalidate of the target address.

If the data is dirty, a clean is performed before the invalidate.

DCISW and DCCSW operations in AArch32, and DC ISW and DC CSW instructions in AArch64 perform both a clean and invalidate of the target set/way. The values of HCR.SWIO and HCR_EL2.SWIO have no effect.

A6.2.5 Data cache coherency

The Cortex-A55 core uses the MESI protocol to maintain data coherency between multiple cores.

MESI describes the state that a shareable line in a L1 data cache can be in:

M Modified/*UniqueDirty* (UD). The line is in only this cache and is dirty.

E Exclusive/*UniqueClean* (UC). The line is in only this cache and is clean.

S Shared/*SharedClean* (SC). The line is possibly in more than one cache and is clean.

I Invalid/*Invalid* (I). The line is not in this cache.

The DCU stores the MESI state of the cache line in the tag and dirty RAMs.

——— **Note** ———

The names UniqueDirty, SharedDirty, UniqueClean, SharedClean, and Invalid are the AMBA names for the cache states. The Cortex-A55 core does not use the SharedDirty AMBA state.

———————————

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 BIU includes logic to detect when the core has written a full cache line before the linefill completes. If this situation is detected on a configurable number of consecutive linefills, then it switches into write streaming mode. This is sometimes referred to as read allocate mode.

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

——— **Note** ———

More than the specified number of linefills might be observed on the ACE or CHI master interface, before the BIU detects that three full cache lines have been written and switches to write streaming mode.

———————————

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

When a CPU has dropped into write streaming mode, the BIU continues to monitor the bus traffic and will signal to the L2 for it to go into write streaming mode when a further number of full cache line writes are seen.

AArch64 state

CPUECTLR_EL1.L1WSCTL configures the L1 write streaming mode threshold,
CPUECTLR_EL1.L2WSCTL configures the L2 write streaming mode threshold, and
CPUECTLR_EL1.L3WSCTL configures the L3 write streaming mode threshold.

AArch32 state

CPUECTLR.L1WSCTL configures the L1 write streaming mode threshold,
CPUECTLR.L2WSCTL configures the L2 write streaming mode threshold, and
CPUECTLR.L3WSCTL configures the L3 write streaming mode threshold.

A6.2.7 Data cache invalidate on reset

The ARMv8-A architecture does not support an operation to invalidate the entire data cache.

The Cortex-A55 core automatically invalidates caches on reset unless suppressed with the debug recovery P-channel state. It is therefore not necessary for software to invalidate the caches on startup.

Related references

Helios specific - ID_MMFR3_EL1, AArch32 Memory Model Feature Register 3, EL1.

Related references

SCTLR_EL1, System Control Register, EL1.

B2.23 CPUACTLR_EL1, CPU Auxiliary Control Register, EL1 on page B2-311.

A6.3 L1 instruction memory system

The L1 instruction side memory system provides an instruction stream to the *Data Processing Unit* (DPU).

To increase overall performance and to reduce power consumption, it uses:

- Dynamic branch prediction.
- Instruction caching.

A6.3.1 Program flow prediction

The Cortex-A55 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 *Virtually Indexed Physically Tagged* (VIPT) cache, with alias avoidance logic so that it appears to software as if it were physically indexed.

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-A55 core supports a single limited order range that includes the entire memory space.

Atomic instructions

The Cortex-A55 core supports the atomic instructions added in 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.

The Cortex-A55 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 a synchronous Data Abort (for load atomics) or an asynchronous Data Abort (for store atomics). The behavior of the atomic instructions can be modified by the CPUECTLR register settings.

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 ARMv8.0 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 loads

Non-temporal loads indicate to the caches that the data is likely to be used for only short periods. For example, when streaming single-use read data that is then discarded. In addition to non-temporal loads, there are also prefetch-memory (`PRFM`) hint instructions with the `STRM` qualifier.

Non-temporal loads cause allocation into the L1 data cache, with the same performance as normal loads. However, when a later linefill is allocated into the cache, the cacheline marked as non-temporal has higher priority to be be replaced. To prevent pollution of the L2 cache, a non-temporal line that is evicted from L1, is not allocated to L2 as would happen for a normal line.

Note

The line is only marked as non-temporal in the cache if the core has the line in a unique state. If shared with other cores, the line is treated normally.

Non-temporal stores are treated as if write streaming mode was active. They are not allocated into any cache in the cluster unless they hit in the cache.

A6.4.2 Internal exclusive monitor

The Cortex-A55 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`.

If a Load-Exclusive instruction is performed to non-cacheable or device memory, and is to a region of memory in the SoC that does not support exclusive accesses, it causes a Data Abort exception with a Data Fault Status Code of either:

- `0b110101`, when using the long descriptor format.
- `0b10101`, when using the short descriptor format.

See the *ARM® Architecture Reference Manual ARMv8, for ARMv8-A architecture profile* for more information about these instructions.

Treatment of intervening STR operations

Where there is an intervening store operation between an exclusive load and an exclusive store from the same core, the intermediate store does not produce any direct effect on the internal exclusive monitor.

After the exclusive load, the local monitor is in the Exclusive Access state. It remains in the Exclusive Access state after the store, and then returns to the Open Access state only after an exclusive store, a `CLREX` instruction, or an exception return.

However, if the exclusive code sequence accessed address is in cacheable memory, any eviction of the cache line containing that address clears the monitor. ARM recommends that no load or store instructions are placed between the exclusive load and the exclusive store, because these additional instructions can cause a cache eviction. Any data cache maintenance instruction can also clear the exclusive monitor.

A6.4.3 Exclusive monitor

In the exclusive state machine, the implementation defined transitions are as follows:

- If the monitor is in the exclusive state, and a store exclusive is performed to a different address, then the store exclusive fails and does not update memory.
- If a normal store is performed to a different address, it does not affect the exclusive monitor.
- If a normal store is performed from a different core to the same address it clears the exclusive monitor. If the store is from the same core then it does not clear the monitor.

A6.5 Data prefetching

The following section describes the software and hardware data prefetching behavior of the Cortex-A55 core.

Hardware data prefetcher

The Cortex-A55 core has a data prefetch mechanism that looks for cache line fetches with regular patterns. If the data prefetcher detects a pattern, then it signals to the memory system that memory accesses from a specified address are likely to occur soon. The memory system responds by starting new linefills to fetch the predicted addresses ahead of the demand loads.

The Cortex-A55 core can track multiple streams in parallel.

Prefetch streams end when either:

- The pattern is broken.
- A `DSB` is executed.
- A WFI or WFE is executed.
- A data cache maintenance operation is executed.

For read streams, the prefetcher is based on the virtual addresses. A given stream is allowed to prefetch addresses through multiple pages as long as they are cacheable and with read permissions. If the new page is still cacheable and has read permission, it can cross page boundaries. Write streams are based on physical addresses and so cannot cross page boundaries. However, if full cache line writes are performed then the prefetcher does not activate and write streaming mode is used instead.

For some types of pattern, when the prefetcher is confident in the stream, it can start progressively increasing the prefetch distance ahead of the current accesses. These accesses start to allocate to the L3 cache rather than L1. Allocating to the L3 cache allows better utilization of the larger resources available at L3. Also, utilizing the L3 cache reduces the amount of pollution of the L1 cache if the stream ends or is incorrectly predicted. If the prefetching to L3 was accurate, the line will be removed from L3 and allocated to L1 when the stream reaches that address.

The *CPU Extended Control Register* (CPUECTLR) allows you to:

- Deactivate the prefetcher.
- Alter the number of outstanding requests that the prefetcher can make.

Preload instructions

The Cortex-A55 core supports `PLD` and `PRFM` instructions. If `PLD` and `PRFM` miss and are to a cacheable address, then these instructions perform a lookup in the cache and start a linefill. The `PRFM`s also enables targeting of a prefetch to the L2 or L3 cache. A request is sent to L2 to start a linefill, and then the instruction can retire without any data being returned to L1. `PLI`,`PLIL1KEEP`, and `PLIL1STRM` are implemented as a prefetch to L2.

Use the `PLD` or `PRFM` instruction for data prefetching where short sequences or irregular pattern fetches are required. For more information about prefetch memory and preloading caches, see the *ARM® Architecture Reference Manual ARMv8, for ARMv8-A architecture profile*.

Data Cache Zero

The Data Cache Zero by Virtual Address (`DC ZVA`) instruction enables a block of 64-bytes in memory, which is aligned to 64-bytes in size, to be set to 0. The DCZID_EL0 register passes this value.

The `DC ZVA` instruction allocates this value into the data cache using the same method as a normal store instruction.

Related references

B2.23 CPUACTLR_EL1, CPU Auxiliary Control Register, EL1 on page B2-311.

A6.6 Direct access to internal memory

The Cortex-A55 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 Cortex-A55 L1 data cache is a 4-way set associative structure.

The size of the configured cache determines the number of sets in each way. The following table shows the encoding (set in `Rd` in the appropriate `MCR` instruction) used to locate the cache data entry for tag and data memory. It is similar for both the tag and data RAM access.

Data RAM access includes an extra field to locate the appropriate word in the cache line. The set-index range parameter (S) is determined by:

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

Table A6-3 Cortex-A55 L1 Data Cache Tag and Data location encoding

Bitfield of Rd	Description
[31:30]	Cache Way
[29:S+5]	Unused
[S+4:6]	Set index
[5:3]	Cache data element offset
[2:0]	Unused (Zero)

Tag information (MESI state, outer attributes, and valid) for the selected cache line, returns using Data Register 0 and Data Register 1.

Use the format that is shown in the following table.

Table A6-4 Cortex-A55 L1 Data Cache Tag data format

Bitfield of Data Register 0 and 1	Description
DR1[31:30]	MESI State (from tag RAM): **0b00** Invalid **0b01** Shared **0b10** Unique non-transient **0b11** Unique transient
DR1[29]	Non-secure state (NS) (from tag RAM)
DR1[28:1]	Tag Address [39:12] (from tag RAM)
DR1[0]	Unused (Zero)
DR0[31:5]	Unused (Zero)
DR0[4]	Dirty bit (from Dirty RAM)
DR0[3]	Shareability (from Dirty RAM)
DR0[2:1]	Age (from Dirty RAM)
DR0[0]	Outer Allocation Hint (from Dirty RAM)

The 64 bits of cache data returns in Data register 0 and Data register 1.

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

The L1 instruction cache is different from the L1 data cache. This is shown in the encodings and data format used in the cache debug operations that are used to access the tag and data memories.

The following table shows the encoding that is required to select a given cache line.

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

$S = \log_2$ (Instruction cache size in bytes/4)

Table A6-5 Cortex-A55 Instruction Cache Tag and Data location encoding

Bit-field of Rd	Description
[31:30]	Cache Way
[29:S+5]	Unused
[S+4:6]	Set index
[5:2]	Cache data element offset (Data Register only)
[1:0]	Unused

The following table shows the tag, instruction, and valid data for the selected cache line using only Data Register.

Table A6-6 Cortex-A55 Instruction Cache Tag data format

Bit-field of Data Register 0	Description	
[31]	Unused	
[30:29]	Valid and set mode:	
	`0b00`	A32
	`0b01`	T32
	`0b10`	A64
	`0b11`	Invalid
[28]	Non-secure state (NS)	-
[27:0]	Tag address	-

The cache data RAMs store instructions in a pre-decoded format. Each A32 or A64 or 32-bit T32 instruction is expanded to 40-bits and each 16-bit T32 instruction occupies 20 bits of the cache. The L1 Instruction Cache Data Read Operation returns two 20-bit entries from the cache in Data Register 0 and Data Register 1. Each corresponds to the 16-bit aligned offset in the cache line:

Data Register 0[19:0] Pre-decode data from cache offset.

Data Register 1[19:0] Pre-decode data from cache offset +2.

In A32 or A64 state, these two combined fields always represent a single pre-decoded instruction. In T32 state, they can represent any combination of 16-bit and partial or full 32-bit instructions.

A6.6.3 Encoding for the L2 TLB

The Cortex-A55 core L2 TLB is built from a 4-way set associative RAM-based structure and contains the data for the main TLB RAM, the Walk cache and IPA cache.

To read the individual entries into the data registers, software must write to the TLB Tag Read Operation Register and to the TLB Data Read Operation Register.

Table A6-7 Cortex-A55 TLB Data Read Operation Register location encoding

Bit-field of Rd	Description
[31:30]	TLB Way
[29:9]	Unused
[8:0]	TLB index

The TLB index is used to select the index from the TLB, walk cache, or IPA cache.

Table A6-8 TLB index

Bit-field of Rd	Description
0x000-0FF	Main TLB
0x100-10F	Walk cache
0x110-11F	IPA cache

The TLB uses an encoding for the descriptor that is returned using the following Data Registers:

Data Register 0[31:0] TLB Descriptor[31:0]
Data Register 1[31:0] TLB Descriptor[63:32]
Data Register 2[31:0] TLB Descriptor[88:64]

A6.6.4 Main TLB RAM descriptor fields

The Main TLB RAM is divided into two parts; one part for storing the tag and the other for storing the data. The following table lists the descriptor fields.

Table A6-9 TLB descriptor fields for Tag RAM

Field	Bits	Width	Description
Valid	[0]	1	Indicates that the entry is valid.
NS (walk)	[1]	1	The security state of core. Used to compare with the NS state for TLB lookup entry match.
ASID	[17:2]	16	Indicates the address space identifier. This field will be 0 if ASID is not used.
VMID	[33:18]	16	Indicates the virtual machine identifier. This field will be 0 if VMID is not used.
Size	[36:34]	3	Indicates the combined page size of stage 1 and stage 2.
nG	[37]	1	Indicates the non-global bit.
AP/HYP	[40:38]	3	AArch32: Access permissions from stage1 translation or select hypervisor mode flag. AArch64: Access permissions from stage 1 translation or select the EL2/EL3 flag.
S2AP	[42:41]	2	Indicates the stage2 permission for EL1/EL0. For EL2/EL3, S2AP[1] is for the stage1 access permission and S2AP[0] is for identify EL2 or EL3.
Domain	[46:43]	4	Indicates the Domain [3:0] information for VMSA and other control information for LPAE.
S1 Size	[49:47]	3	Indicates the page or block size of the stage 1 translation result.
Address Sign bit	[50]	1	Indicates the VA sign bit, VA[48].

Table A6-9 TLB descriptor fields for Tag RAM (continued)

Field	Bits	Width	Description
VA	[78:51]	28	Indicates the virtual address.
DBM	[79]	1	Indicates the *Dirty Bit Modifier* (DBM) bit.
Parity	[81:80]	2	Indicates the parity bits. If parity is not configured, their bits are absent.

Table A6-10 TLB descriptor fields for Data RAM

Field	Bits	Width	Description
XS1Usr	[0]	1	AArch32: Executable and Readable in stage1 user mode. AArch64: Executable in stage1 user mode
XS1Non-Usr	[1]	1	AArch32 and AArch 64: Executable in stage 1 non-user mode.
XS2Usr	[2]	1	AArch32 and AArch 64: Executable in stage 2 user mode.
XS2Non-Usr	[3]	1	AArch32 and AArch 64: Executable in stage 2 non-user mode.
Memory type and shareability	[11:4]	8	Defines the memory attribute.
S2 Level	[13:12]	2	The stage 2 level that gave this translation.
NS (descriptor)	[14]	1	The security state allocated to this memory region.
PA	[42:15]	28	The physical address.
Parity	[43]	1	Parity inclusion is dependant on configuration.

A6.6.5 Walk cache descriptor fields

The following table shows the walk cache descriptor data fields for Tag and Data RAMs.

Table A6-11 Walk cache descriptor fields for Tag RAM

Field	Bit Position	Width	Description
Valid	[0]	1	Indicates that the entry is valid.
NS (walk)	[1]	1	The Security state of the entry fetch.
ASID	[17:2]	16	Address Space Identifier.
VMID	[33:18]	16	Virtual Machine Identifier.
HYP/EL2	[34]	1	Set if the entry was fetched in HYP, EL2, or *Virtual Host Extension* (VHE) mode.
EL3	[35]	1	Set if the entry was fetched in AArch64 EL3 mode.
Arch	[38:36]	3	Used to determine how many and which bits of address are used for constructing the physical address of the pagewalk.
Domain	[42:39]	4	Valid only if the entry was fetched in VMSAv7 format.
Address Sign Bit	[45]	1	Address sign bit, VA[48].
VA	[69:46]	24	Virtual Address sign bit.
S2AP	[71:70]	2	Stage 2 access permission.
S2level	[73:72]	2	The stage 2 level which translates the IPA to PA for the page table entry.
Parity	[81:80]	2	Parity Bits.

Table A6-12 Walk cache descriptor fields for Data RAM

Field	Bit Position	Width	Description
APTable	[1:0]	2	Combined ATable bits from stage 1 descriptors up to the last level.
XNTable	[2]	1	Combined XNTAble bits from stage 1 descriptors up to the last level.
PXNTable	[3]	1	Combined PXNTable bits from stage 1 descriptors up to the last level.
NSTable	[4]	1	Combined NSTable bits from first and second-level stage 1 tables or NS descriptors (VMSA).
Attrs	[12:5]	8	Physical address attributes of the final level stage 1 table.
PA	[42:13]	30	Physical address of the stage 1 last translation level page table entry.
Parity	[43]	1	Parity inclusion is core configuration dependent. If parity is no configured, these bits are absent.

A6.6.6 IPA cache descriptor fields

The IPA cache holds mappings from intermediate physical addresses (IPA) to physical addresses. It is only used for translations performed in non-secure EL0/1. It is updated whenever a stage 2 translation is completed, and checked whenever a stage 2 translation is required.

The following table shows the data and tag fields in the IPA cache descriptor.

Table A6-13 IPA cache descriptor fields for Tag RAM

Fields	Bits	Width	Descriptor
Valid	[0]	1	Indicates that the entry is valid.
Entry granule	[2:1]	2	Indicates the entry granule size.
Unused	[4:3]	2	Must be set to 0.
Size	[8:5]	4	Indicates the S2 page size for this entry.
DBM	[9]	1	Indicates the DBM.
Unused	[17:10]	8	Must be set to 0.
VMID	[33:18]	16	Indicates the virtual machine identifier.
IPA	[57:34]	24	Unused lower bits, page size dependant, must be set to zero.
Unused	[79:59]	22	Must be set to zero.
Parity	[81:80]	2	If parity is not configured, this bit is absent.

Table A6-14 IPA cache descriptor fields for Data RAM

Fields	Bits	Width	Descriptor
SH	[1:0]	2	Shareability.
S2AP	[3:2	2	Stage 2 access permissions
XN	[5:4]	2	Controls EL1 and EL0 access permissions.
Memattrs	[9:6]	4	Stage 2 memory attributes.
PA	[37:10]	28	Physical Address.
Unused	[42:38]	5	Must be set to zero.
Parity	[43]	1	If parity is not configured, this bit is absent.

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 Cortex-A55 L2 memory system is required to interface the Cortex-A55 cores to the L3 memory system.

The L2 cache controller handles requests from the L1 instruction and data caches, and snoop requests from the L3 memory system. The L2 memory system forwards responses from the L3 system to the core, which can then take precise or imprecise aborts, depending on the type of transaction.

The L2 memory subsystem consists of:

- An optional 4-way, set-associative L2 cache with a configurable size of 64KB, 128KB or 256KB. Cache lines have a fixed length of 64 bytes.
- Optional ECC protection for tag, data, and L2 data buffer RAM structures.

The main features of the L2 memory system are:

- Strictly exclusive with L1 data cache.
- Pseudo-inclusive with L1 instruction cache.
- Private per-core unified L2 cache.
- 40-bit physical address space.
- Physically indexed, physically tagged.

A7.2 Optional integrated L2 cache

Data is allocated to the L2 cache only when evicted from the L1 memory system, not when first fetched from the system.

The exceptions to this rule are:

- If the read-allocate hint is set, cacheable reads from the TLB or *Instruction side* (I-side) will be allocated in the L2 cache.
- If the write-allocate hint is set, when the L1 enters write-streaming mode, cacheable writes will be allocated in the L2.
- L2 prefetches issued by the L1 through a `PLD` or `PRFM` instruction are allocated in the L2 regardless of the read-allocate hint.

When non-temporal data is evicted from the L1 memory system, the data is sent directly to L3 and is not allocated in L2.

L2 RAMs are invalidated automatically at reset unless the debug recovery P-channel state is used.

A7.3 Support for memory types

The Cortex-A55 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.
- All other memory types are Non-cacheable.

The additional attribute hints are used as follows:

Allocation hint

Determines the rules of allocation of newly fetched lines in the system, see *A7.2 Optional integrated L2 cache* on page A7-91.

Transient hint

Allocating reads (from TLB or I-side) and writes in write-streaming mode that have the transient bit set are allocated in L2 cache and marked as most likely to be evicted according to the L2 eviction policy.

Evictions from L1 cache marked as transient are not allocated in L2 cache.

The standard CHI attributes are passed to DSU with no modifications (except for translating architectural attributes to CHI attributes):

- Allocate hint.
- Cacheability (inner and outer are merged together, as the Cortex-A55 core only allocates both inner and outer cacheable memory).
- Shareability.

Chapter A8
Reliability, Availability, and Serviceability (RAS)

This chapter describes the RAS features implemented in the Cortex-A55 core.

It contains the following sections:

A8.1 Cache ECC and parity

The Cortex-A55 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-A55 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 core protects against soft errors that result in a RAM bitcell temporarily holding the incorrect value.

The Cortex-A55 core writes a new value to the RAM to correct the error. If the error is a hard error that is not corrected by writing to the RAM, for example a physical defect in the RAM, then the core might get into a livelock as it continually detects and then tries to correct the error.

Some RAMs have *Single Error Detect* (SED) capability, while others have *Single Error Correct, Double Error Detect* (SECDED) capability. The core can make progress and remain functionally correct when there is transient 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 listed in the following table, the error is detected and reported as described in error reporting. If the error is in a cache line containing dirty data, then that data might be lost, resulting in data corruption
- For RAMs with only SED, a double bit error is not detected and therefore might cause data corruption.

If there are three or more bit errors, then depending on the RAM and the position of the errors within the RAM, the errors might be detected or might not be detected.

The Cortex-A55 cache protection support has a minimal performance impact when no errors are present. When an error is detected, the access that caused the error is stalled while the correction takes place. When the correction is complete, the access either continues with the corrected data, or is retried. If the access is retried, it either hits in the cache again with the corrected data, or misses in the cache and re-fetches the data from a lower level cache or from main memory. The behavior for each RAM is shown in the following table.

Table A8-1 Cache protection behavior

RAM	Protection type	Protection granule	Correction behavior
L1 instruction cache tag	Parity, SED	31 bits	Both lines in the cache set are invalidated, then the line requested is refetched from L2 or external memory.
L1 instruction cache data	Parity, SED	20 bits	Both lines in the cache set are invalidated, then the line requested is refetched from L2 or external memory.
L2 TLB tag	Parity, SED	39 bits or 40 bits	Entry invalidated, new pagewalk started to refetch it.
L2 TLB data	Parity, SED	43 bits	Entry invalidated, new pagewalk started to refetch it.
L1 data cache tag	SECDED	32 bits	Line cleaned and invalidated from L1. SCU duplicate tags are used to get the correct address. Line refetched from L2 or external memory, with single bit errors corrected as part of the eviction.
L1 data cache data	ECC, SECDED	32 bits	Line cleaned and invalidated from L1, with single bit errors corrected as part of the eviction. Line refetched from L2 or external memory.
L1 data cache dirty	ECC, SECDED	2 bits	Line cleaned and invalidated from L1, with single bit errors corrected as part of the eviction. Only the dirty bit is protected. The other bits are performance hints, therefore do not cause a functional failure if they are incorrect.
L2 cache tag	ECC, SECDED	30, 31, or 32 bits depending on the cache size.	Tag rewritten with correct value, access retried. If the error is uncorrectable then the tag is invalidated.

Table A8-1 Cache protection behavior (continued)

RAM	Protection type	Protection granule	Correction behavior
L2 cache victim	None	-	The victim RAM is used only as a performance hint. It does not result in a functional failure if the contents are incorrect.
L2 cache data	ECC, SECDED	64 bits	Data is corrected inline, access might stall for an additional cycle or two while the correction takes place.
L2 data buffer	ECC, SECDED	72 bits	Data is corrected inline, access might stall for an additional cycle or two while the correction takes place.
Branch predictor	None	-	The branch predictor RAMs are used only as a performance hint. They do not result in a functional failure if the contents are incorrect.

Note

When an ECC error occurs during a load instruction that takes multiple cycles to complete, for example `LDM`, the load instruction will re-execute. However, if a state change occurs between the original load instruction and the second attempt, then the second attempt will not execute. The first attempt could leave the register file in an inconsistent state, since the register file may have been updated for locations that did not have errors.
The following situations will cause a state change between the original instruction and the second attempt:

- A hardware breakpoint, watchpoint, or vector catch has been set since the first execution that is triggered on re-execution.
- The page tables have been modified since the first execution, resulting in an instruction or data abort trap being taken on re-execution.

In these situations, software may be able to observe that the original load instruction committed some new state despite not fully completing.

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 back into a cache. The poisoned data is stored per 64 bits of data, except in the L1 data cache where it is stored per 32 bits of data.
- If the interconnect supports poisoning, then the poison is passed along with the data when the line is evicted from the cluster. No abort is generated when a line is poisoned, as the abort can be deferred until the point when the poisoned data is consumed by a load or instruction fetch.

Uncorrected error detected in a tag or dirty RAM

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

A8.4 RAS error types

For a standard error record, three error types can be recorded.

When a processing element accesses memory or other state, errors might be detected in that memory or state, and corrected, deferred, or signaled to the processing element as a detected error. The component that detects an error is called a node.

Corrected error (CE) An error was detected and corrected. It no longer infects the state of the node and has not been silently propagated. The node continues to operate.

Deferred error (DE) An error was detected, was not corrected, and was deferred. The error is not silently propagated and may be latent in the system. The node continues to operate.

Uncorrected Error (UC) An error was detected and was not corrected or deferred. The error is latent in the system.

——— **Note** ———

Uncorrected errors can have subtypes depending on whether the error was produced or consumed at the node.

———

Errors produced at the node

For uncorrected errors that are produced at the node, the subtypes, in increasing severity, are:

Latent The error has not been propagated. That is, the error was detected but not consumed, and was not recorded as a deferred error.

Signaled The error has not been silently propagated. The error has been or might have been consumed, and was not recorded as a deferred error.

——— **Note** ———

The producer cannot know if a consumer has architecturally consumed the error. If it has definitely not been propagated to any consumer, and signaled otherwise, an error might be marked as Latent.

———

Unrecoverable (UEU) The error has not been silently propagated. The node cannot continue operating.

Uncontainable (UC) The error might have been silently propagated. If the error cannot be isolated, the system must be shut down to avoid catastrophic failure.

Errors consumed at the node

For uncorrected errors that are consumed at the node, the subtypes, in increasing severity, are:

Restartable (UEO) The error has not been silently propagated. The node halts operation because of consuming an error. The node does not rely on the corrupted data so can continue to operate without repairing the error.

Recoverable (UER) The error has not been silently propagated. The node halts operation. To continue, the node relies on consuming the corrupted data. If software can locate and repair the error, the halted operation can continue.

Unrecoverable (UEU) The error has not been silently propagated. The node halts operation and cannot resume from its halted state.

Uncontainable (UC) The error might have been silently propagated. If the error cannot be isolated, the system must be shut down to avoid catastrophic failure.

Error status priority

The highest priority recorded error type is recorded in the Error Record Primary Syndrome Register.

Related references

ERXSTATUS, Error Record Primary Syndrome Register.

A8.5 Error synchronization barrier

The *Error Synchronization Barrier* (ESB) instruction synchronizes unrecoverable errors.

The RAS extension adds the ESB instruction used to synchronize unrecoverable errors. Unrecoverable errors are containable errors consumed by the core and not silently propagated.

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 unrecoverable errors must be synchronized by an ESB instruction. The ESB instruction guarantees the following:

- All unrecoverable 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 is executed:
  - If the physical SEI is unmasked at the current Exception level, then it is taken before completion of the ESB instruction.
  - If the physical SEI is masked at the current Exception level, the pending SEI is cleared, the SEI syndrome is recorded in DISR/DISR_EL1, and DISR/DISR_EL1.A is set to 1. This indicates that the SEI was generated before the ESB by instructions that occur in programme order.

The ESB instruction also guarantees the following:

- SEIs generated before the ESB instruction are either taken before or at the ESB instruction, or are pended in DISR/DISR_EL1.
- SEIs generated after the ESB are not pended in DISR/DISR_EL1.

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

Note

DISR/DISR_EL1 can only be accessed at EL1 or 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 reporting

Detected errors are reported in the Error Record Primary Syndrome Register, ERXSTATUS/ERXSTATUS_EL1, and the Error Record Miscellaneous Register, ERXMISC0/ERXMISC0_EL1.

This includes errors that are successfully corrected, and errors that cannot be corrected. If multiple errors occur on the same clock cycle, then only one error is reported but the OF (overflow) bit is set.

There are two error records provided, which can be selected with the ERRSELR/ERRSELR_EL1 register. Record 0 is private to the core, and is updated on any error in the 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 cores in the cluster.

If enabled in the ERXCTLR/ERXCTLR_EL1 register, all errors that are detected cause a fault handling interrupt. 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.

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.

- Uncorrectable errors in the L1, L2, or L3 data RAMs when read by an instruction fetch, a load instruction or a TLB pagewalk, might result in a precise data abort or prefetch abort.
- Uncorrectable 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. If the poisoned line is evicted from the cluster for any reason, and the interconnect does not support data poisoning, then the **nERRIRQ[0]** pin is asserted, if enabled.
- Uncorrectable errors in the L1 tag or dirty RAMs, or in the L2 tag RAMs, causes the **nERRIRQ[n+1]** pin to be asserted for core *n*, if enabled.
- Uncorrectable errors in the L3 tag RAMs or SCU snoop filter RAMs causes the **nERRIRQ[0]** pin to be asserted, if enabled.

Note

- When **nERRIRQ** is asserted it remains asserted until the error is cleared by a write of 0 to the UE bit in the ERXSTATUS/ERXSTATUS_EL1 register.
- ARM recommends that the **nERRIRQ** pin is connected to the interrupt controller so that an interrupt or system error is generated when the pin is asserted.

The fault and error interrupt pins are cleared by writing to the ERXSTATUS/ERXSTATUS_EL1 registers.

When a snoop hits on a line with an uncorrectable data error, the data is returned if required by the snoop, but the snoop response indicates that the data is poisoned. If a snoop hits on a tag that has an uncorrectable error, then it is treated as a snoop miss, because the error means that it is unknown if the cache line is valid or not.

The following accesses update the Error Record Primary Syndrome Register:

- ECC error detected in any of the RAM protected by ECC.
- Poisoned data received from the DSU when the CPU does not support ECC protection.
- Dirty data received from the DSU and the data is flagged with a data error.

Note

It is possible for an error to be counted 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 ERRXSTATUS and ERRXMISCn registers:

- If two or more memory errors occur in the same cycle, only one error is reported and the other error count is incremented. If more than two errors occur on the same cycle then the additional errors will not be counted.
- If two or more first memory error events from different RAMs occur in the same cycle, one of the errors is selected arbitrarily.
- If a new error arrives while the ERRXSTATUS.V bit is set, the way, index, and level information is not updated, but the other error field or the repeat error field is updated.
- If two or more memory errors from different RAMs that do not match the level, way and index information in this register when the ERRXSTATUS.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.

Related references

C2.4 PMU events on page C2-531.

ERXSTATUS, Error Record Primary Syndrome Register.

A8.7 Error injection

To support testing of error handling software, the Cortex-A55 core can fake errors in the error detection logic.

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

Table A8-2 Errors injected in the Cortex-A55 core

Error type	Description
Corrected errors	A CE is generated for a single ECC error on L1 data cache access.
Deferred errors	A DE is generated for a double ECC error on eviction of a cache line from the L1 to the L2, or as a result of a snoop on the L1.
Uncontainable errors	A UC is generated for a double ECC error on the L1 TAG RAM following an eviction.
Latent error	A UEO is generated as a double ECC error on an L1 data read.

The following table describes the registers that handle error injection in the Cortex-A55 core.

Table A8-3 Error injection registers

Register name	Description
ERR<n>PFGFR	The ERR Pseudo Fault Generation Feature register defines which errors can be injected.
ERR<n>PFGCTLR	The ERR Pseudo Fault Generation Control register controls the errors that are injected.
ERXPFGCDN_EL1	The Selected 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 corrupted.

Chapter A9
Generic Interrupt Controller CPU interface

This chapter describes the Cortex-A55 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-A55 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-A55 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-A55 core implementation.

A9.2 Bypassing the CPU Interface

The GIC CPU Interface is always implemented within the Cortex-A55 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-A55 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.

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-A55 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-A55 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-A55 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-A55 core.

The section contains two tables:

Registers with implementation defined bit fields

This table identifies the architecturally defined registers in the Cortex-A55 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 B1-115.

Other architecturally defined registers

This table identifies the other architecturally defined registers that are implemented in the Cortex-A55 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 B1-118.

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 B1-128
ACTLR2	c1	0	c0	3	32	*B1.6 ACTLR2, Auxiliary Control Register 2* on page B1-130
AIDR	c0	1	c0	7	32	*B1.8 AIDR, Auxiliary ID Register* on page B1-132
ADFSR	c5	0	c1	0	32	*B1.7 ADFSR, Auxiliary Data Fault Status Register* on page B1-131
AIFSR	c5	0	c1	1	32	*B1.9 AIFSR, Auxiliary Instruction Fault Status Register* on page B1-133
AMAIR0	c10	0	c3	0	32	*B1.10 AMAIR0, Auxiliary Memory Attribute Indirection Register 0* on page B1-134
AMAIR1	c10	0	c3	1	32	*B1.11 AMAIR1, Auxiliary Memory Attribute Indirection Register 1* on page B1-135
CCSIDR	c0	1	c0	0	32	*B1.12 CCSIDR, Cache Size ID Register* on page B1-136
CLIDR	c0	1	c0	1	32	*B1.13 CLIDR, Cache Level ID Register* on page B1-139
CPACR	c1	0	c0	2	32	*B1.14 CPACR, Architectural Feature Access Control Register* on page B1-141
CSSELR	c0	2	c0	0	32	*B1.23 CSSELR, Cache Size Selection Register* on page B1-159
CTR	c0	0	c0	1	32	*B1.24 CTR, Cache Type Register* on page B1-160
DFSR	c5	0	c0	0	32	*B1.25 DFSR, Data Fault Status Register* on page B1-162
DISR	c12	0	c1	1	32	*B1.26 DISR, Deferred Interrupt Status Register* on page B1-164
ERRIDR	c5	0	c3	0	32	*B1.27 ERRIDR, Error ID Register* on page B1-167
ERRSELR	c5	0	c3	1	32	*B1.28 ERRSELR, Error Record Select Register* on page B1-168
ERXADDR	c5	0	c4	3	32	*B1.29 ERXADDR, Selected Error Record Address Register* on page B1-169

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

Name	CRn	Opc1	CRm	Opc2	Width	Description
ERXADDR2	c5	0	c4	7	32	*B1.30 ERXADDR2, Selected Error Record Address Register 2* on page B1-170
ERXCTLR	c5	0	c4	1	32	*B1.31 ERXCTLR, Selected Error Record Control Register* on page B1-171
ERXCTLR2	c5	0	c4	5	32	*B1.32 ERXCTLR2, Selected Error Record Control Register 2* on page B1-172
ERXFR	c5	0	c4	0	32	*B1.33 ERXFR, Selected Error Record Feature Register* on page B1-173
ERXFR2	c5	0	c4	4	32	*B1.34 ERXFR2, Selected Error Record Feature Register 2* on page B1-174
ERXMISC0	c5	0	c5	0	32	*B1.35 ERXMISC0, Selected Error Miscellaneous Register 0* on page B1-175
ERXMISC1	c5	0	c5	1	32	*B1.36 ERXMISC1, Selected Error Miscellaneous Register 1* on page B1-176
ERXMISC2	c5	0	c5	4	32	*B1.37 ERXMISC2, Selected Error Record Miscellaneous Register 2* on page B1-177
ERXMISC3	c5	0	c5	5	32	*B1.38 ERXMISC3, Selected Error Record Miscellaneous Register 3* on page B1-178
ERXSTATUS	c5	0	c4	2	32	*B1.42 ERXSTATUS, Selected Error Record Primary Status Register* on page B1-184
FCSEIDR	c13	0	c0	0	32	*B1.43 FCSEIDR, FCSE Process ID Register* on page B1-185
FPSID	c5	4	c3	1	32	Floating-Point System ID Register
HACR	c1	4	c1	7	32	*B1.44 HACR, Hyp Auxiliary Configuration Register* on page B1-186
HACTLR	c1	4	c0	1	32	*B1.45 HACTLR, Hyp Auxiliary Control Register* on page B1-187
HACTLR2	c1	4	c0	3	32	*B1.46 HACTLR2, Hyp Auxiliary Control Register 2* on page B1-189
HADFSR	c5	4	c1	0	32	*B1.47 HADFSR, Hyp Auxiliary Data Fault Status Syndrome Register* on page B1-190
HAIFSR	c5	4	c1	1	32	*B1.48 HAIFSR, Hyp Auxiliary Instruction Fault Status Syndrome Register* on page B1-191
HAMAIR0	c10	4	c3	0	32	*B1.49 HAMAIR0, Hyp Auxiliary Memory Attribute Indirection Register 0* on page B1-192
HAMAIR1	c10	4	c3	1	32	*B1.50 HAMAIR1, Hyp Auxiliary Memory Attribute Indirection Register 1* on page B1-193
HCR	c1	4	c1	0	32	*B1.51 HCR, Hyp Configuration Register* on page B1-194
HCR2	c1	4	c1	4	32	*B1.52 HCR2, Hyp Configuration Register 2* on page B1-196
HSR	c5	4	c2	0	32	*B1.54 HSR, Hyp Syndrome Register* on page B1-199
ID_AFR0	c0	0	c1	3	32	*B1.56 ID_AFR0, Auxiliary Feature Register 0* on page B1-202
ID_DFR0	c0	0	c1	2	32	*B1.57 ID_DFR0, Debug Feature Register 0* on page B1-203
ID_ISAR0	c0	0	c2	0	32	*B1.58 ID_ISAR0, Instruction Set Attribute Register 0* on page B1-205

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

Name	CRn	Opc1	CRm	Opc2	Width	Description
ID_ISAR1	c0	0	c2	1	32	*B1.59 ID_ISAR1, Instruction Set Attribute Register 1* on page B1-207
ID_ISAR2	c0	0	c2	2	32	*B1.60 ID_ISAR2, Instruction Set Attribute Register 2* on page B1-209
ID_ISAR3	c0	0	c2	3	32	*B1.61 ID_ISAR3, Instruction Set Attribute Register 3* on page B1-211
ID_ISAR4	c0	0	c2	4	32	*B1.62 ID_ISAR4, Instruction Set Attribute Register 4* on page B1-213
ID_ISAR5	c0	0	c2	5	32	*B1.63 ID_ISAR5, Instruction Set Attribute Register 5* on page B1-215
ID_ISAR6	c0	0	c2	7	32	*B1.64 ID_ISAR6, Instruction Set Attribute Register 6* on page B1-217
ID_MMFR0	c0	0	c1	4	32	*B1.65 ID_MMFR0, Memory Model Feature Register 0* on page B1-218
ID_MMFR1	c0	0	c1	5	32	*B1.66 ID_MMFR1, Memory Model Feature Register 1* on page B1-220
ID_MMFR2	c0	0	c1	6	32	*B1.67 ID_MMFR2, Memory Model Feature Register 2* on page B1-222
ID_MMFR3	c0	0	c1	7	32	*B1.68 ID_MMFR3, Memory Model Feature Register 3* on page B1-224
ID_MMFR4	c0	0	c2	6	32	*B1.69 ID_MMFR4, Memory Model Feature Register 4* on page B1-226
ID_PFR0	c0	0	c1	0	32	*B1.70 ID_PFR0, Processor Feature Register 0* on page B1-228
ID_PFR1	c0	0	c1	1	32	*B1.71 ID_PFR1, Processor Feature Register 1* on page B1-229
IFSR	c5	0	c0	1	32	*B1.72 IFSR, Instruction Fault Status Register* on page B1-231
MIDR	c0	0	c0	0, 4, or 7	32	*B1.73 MIDR, Main ID Register* on page B1-233
MPIDR	c0	0	c0	5	32	*B1.74 MPIDR, Multiprocessor Affinity Register* on page B1-234
PAR (32 bits access)	c7	0	c4	0	32	*B1.75 PAR, Physical Address Register* on page B1-236
PAR (64 bits access)	-	0	c7	-	64	*B1.75 PAR, Physical Address Register* on page B1-236
REVIDR	c0	0	c0	6	32	*B1.76 REVIDR, Revision ID Register* on page B1-241
SCR	c1	0	c1	0	32	*B1.77 SCR, Secure Configuration Register* on page B1-242
SCTLR	c1	0	c0	0	32	*B1.78 SCTLR, System Control Register* on page B1-243
SDCR	c1	0	c3	1	32	*B1.79 SDCR, Secure Debug Control Register* on page B1-246
TTBCR	c2	0	c0	2	32	*B1.80 TTBCR, Translation Table Base Control Register* on page B1-248
TTBR0 (32 bits access)	c2	0	c0	0	32	*B1.82 TTBR0, Translation Table Base Register 0* on page B1-255
TTBR0 (64 bits access)	-	0	c2	-	64	*B1.82 TTBR0, Translation Table Base Register 0* on page B1-255

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	*B1.83 TTBR1, Translation Table Base Register 1* on page B1-257
TTBR1 (64 bits access)	-	1	c2	-	64	*B1.83 TTBR1, Translation Table Base Register 1* on page B1-257
VDFSR	c5	4	c2	3	32	*B1.84 VDFSR, Virtual SError Exception Syndrome Register* on page B1-259
VDISR	c12	4	c1	1	32	*B1.85 VDISR, Virtual Deferred Interrupt Status Register* on page B1-260
VMPIDR	c0	4	c0	5	32	*B1.86 VMPIDR, Virtualization Multiprocessor ID Register* on page B1-263
VPIDR	c0	4	c0	0	32	*B1.87 VPIDR, Virtualization Processor ID Register* on page B1-264
VTCR	c2	4	c1	2	32	*B1.88 VTCR, Virtualization Translation Control Register* on page B1-265
VTTBR	-	6	c2	-	64	*B1.89 VTTBR, Virtualization Translation Table Base Register* on page B1-267

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-A55 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-A55 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 B1-142
CPUCFR	cp15	c15	0	c0	0	32	*B1.16 CPUCFR, CPU Configuration Register* on page B1-144
CPUECTLR	cp15	-	4	c15	-	64	*B1.17 CPUECTLR, CPU Extended Control Register* on page B1-146
CPUPCR	cp15	c15	6	c8	1	64	*B1.18 CPUPCR, CPU Private Control Register* on page B1-149
CPUPMR	cp15	c15	6	c8	3	64	*B1.19 CPUPMR, CPU Private Mask Register* on page B1-151
CPUPOR	cp15	c15	6	c8	2	64	*B1.20 CPUPOR, CPU Private Operation Register* on page B1-153
CPUPSELR	cp15	c15	6	c8	0	32	*B1.21 CPUPSELR, CPU Private Selection Register* on page B1-155
CPUPWRCTLR	cp15	c15	0	c2	7	32	*B1.22 CPUPWRCTLR, CPU Power Control Register* on page B1-157
ERR0PFGFR	N/A	N/A	N/A	N/A	N/A	32	*B3.9 ERR0PFGFR, Error Pseudo Fault Generation Feature Register* on page B3-447
ERR0PFGCTLR	N/A	N/A	N/A	N/A	N/A	32	*B3.8 ERR0PFGCTLR, Error Pseudo Fault Generation Control Register* on page B3-445
ERR0PFGCDNR	N/A	N/A	N/A	N/A	N/A	32	*B3.7 ERR0PFGCDNR, Error Pseudo Fault Generation Count Down Register* on page B3-444
ERXPFGCDNR	cp15	c15	0	c2	2	32	*B1.39 ERXPFGCDNR, Selected Error Pseudo Fault Generation Count Down Register* on page B1-179
ERXPFGCTLR	cp15	c15	0	c2	1	32	*B1.40 ERXPFGCTLR, Selected Error Pseudo Fault Generation Control Register* on page B1-181
ERXPFGFR	cp15	c15	0	c2	0	32	*B1.41 ERXPFGFR, Selected Pseudo Fault Generation Feature Register* on page B1-183

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.
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.

Table B1-4 Cluster registers (continued)

Name	Copro	CRn	Opc1	CRm	Opc2	Width	Description
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 B1-132
CCSIDR	RO	-	*B1.12 CCSIDR, Cache Size ID Register* on page B1-136
CLIDR	RO	UNK Unknown: [31:30], [25:24], [22:21], 8, 5 'b1: [1:0] 'b0: [29:26], 23, [20:9], [7:6], [4:2]	*B1.13 CLIDR, Cache Level ID Register* on page B1-139 If the L2 cache is not implemented, the value is: `0x09200003`
CSSELR	RW	`0x00000000`, [2:0] UNK.	*B1.23 CSSELR, Cache Size Selection Register* on page B1-159
CTR	RO	`0x84448004`	*B1.24 CTR, Cache Type Register* on page B1-160
ID_AFR0	RO	`0x00000000`	*B1.56 ID_AFR0, Auxiliary Feature Register 0* on page B1-202
ID_DFR0	RO	`0x03010066`	*B1.57 ID_DFR0, Debug Feature Register 0* on page B1-203 Bits [19:16] are `0x1` if ETM is implemented, and `0x0` otherwise.
ID_ISAR0	RO	`0x02101110`	*B1.58 ID_ISAR0, Instruction Set Attribute Register 0* on page B1-205
ID_ISAR1	RO	`0x13112111`	*B1.59 ID_ISAR1, Instruction Set Attribute Register 1* on page B1-207
ID_ISAR2	RO	`0x21232042`	*B1.60 ID_ISAR2, Instruction Set Attribute Register 2* on page B1-209
ID_ISAR3	RO	`0x01112131`	*B1.61 ID_ISAR3, Instruction Set Attribute Register 3* on page B1-211
ID_ISAR4	RO	`0x00011142`	*B1.62 ID_ISAR4, Instruction Set Attribute Register 4* on page B1-213
ID_ISAR5	RO	`0x00011121`	*B1.63 ID_ISAR5, Instruction Set Attribute Register 5* on page B1-215 ID_ISAR5 has the value `0x00010001` if the Cryptographic Extension is not implemented and enabled.
ID_ISAR6	RO	`0x00000010`	*B1.64 ID_ISAR6, Instruction Set Attribute Register 6* on page B1-217
ID_MMFR0	RO	`0x10201105`	*B1.65 ID_MMFR0, Memory Model Feature Register 0* on page B1-218
ID_MMFR1	RO	`0x40000000`	*B1.66 ID_MMFR1, Memory Model Feature Register 1* on page B1-220
ID_MMFR2	RO	`0x01260000`	*B1.67 ID_MMFR2, Memory Model Feature Register 2* on page B1-222
ID_MMFR3	RO	`0x02102211`	*B1.68 ID_MMFR3, Memory Model Feature Register 3* on page B1-224
ID_MMFR4	RO	`0x00021110`	*B1.69 ID_MMFR4, Memory Model Feature Register 4* on page B1-226
ID_PFR0	RO	`0x00000131`	*B1.70 ID_PFR0, Processor Feature Register 0* on page B1-228
ID_PFR1	RO	`0x10011011`	*B1.71 ID_PFR1, Processor Feature Register 1* on page B1-229 Bits [31:28] are `0x1` if the GIC CPU interface is implemented and enabled, and `0x0` otherwise.

(continued)

Name	Type	Reset	Description
MIDR	RO	0x411FD050	*B1.73 MIDR, Main ID Register* on page B1-233
MPIDR	RO	-	*B1.74 MPIDR, Multiprocessor Affinity Register* on page B1-234
REVIDR	RO	0x00000000	*B1.76 REVIDR, Revision ID Register* on page B1-241
VMPIDR	RW	-	*B1.86 VMPIDR, Virtualization Multiprocessor ID Register* on page B1-263 The reset value is the value of MPIDR.
VPIDR	RW	0x411FD050	*B1.87 VPIDR, Virtualization Processor ID Register* on page B1-264

Other system control registers

Name	Type	Description
ACTLR	RW	*B1.5 ACTLR, Auxiliary Control Register* on page B1-128
ACTLR2	RW	*B1.6 ACTLR2, Auxiliary Control Register 2* on page B1-130
HACTLR	RW	*B1.45 HACTLR, Hyp Auxiliary Control Register* on page B1-187
HACTLR2	RW	*B1.46 HACTLR2, Hyp Auxiliary Control Register 2* on page B1-189
HSCTLR	RW	*B1.53 HSCTLR, Hyp System Control Register* on page B1-197
SCTLR	RW	*B1.78 SCTLR, System Control Register* on page B1-243

RAS registers

Name	Type	Description
DISR	RW	*B1.26 DISR, Deferred Interrupt Status Register* on page B1-164
ERRIDR	RO	*B1.27 ERRIDR, Error ID Register* on page B1-167
ERRSELR	RW	*B1.28 ERRSELR, Error Record Select Register* on page B1-168
ERXADDR	RW	*B1.29 ERXADDR, Selected Error Record Address Register* on page B1-169
ERXADDR2	RW	*B1.30 ERXADDR2, Selected Error Record Address Register 2* on page B1-170
ERXCTLR	RW	*B1.31 ERXCTLR, Selected Error Record Control Register* on page B1-171
ERXCTLR2	RW	*B1.32 ERXCTLR2, Selected Error Record Control Register 2* on page B1-172
ERXFR	RO	*B1.33 ERXFR, Selected Error Record Feature Register* on page B1-173
ERXFR2	RO	*B1.34 ERXFR2, Selected Error Record Feature Register 2* on page B1-174
ERXMISC0	RW	*B1.35 ERXMISC0, Selected Error Miscellaneous Register 0* on page B1-175
ERXMISC1	RW	*B1.36 ERXMISC1, Selected Error Miscellaneous Register 1* on page B1-176
ERXMISC2	RW	*B1.37 ERXMISC2, Selected Error Record Miscellaneous Register 2* on page B1-177
ERXMISC3	RW	*B1.38 ERXMISC3, Selected Error Record Miscellaneous Register 3* on page B1-178
ERXPFGCDNR	RW	*B1.39 ERXPFGCDNR, Selected Error Pseudo Fault Generation Count Down Register* on page B1-179
ERXPFGCTLR	RW	*B1.40 ERXPFGCTLR, Selected Error Pseudo Fault Generation Control Register* on page B1-181
ERXPFGFR	RO	*B1.41 ERXPFGFR, Selected Pseudo Fault Generation Feature Register* on page B1-183
ERXSTATUS	RW	*B1.42 ERXSTATUS, Selected Error Record Primary Status Register* on page B1-184
HCR2	RW	*B1.52 HCR2, Hyp Configuration Register 2* on page B1-196

(continued)

Name	Type	Description
VDFSR	RW	*B1.84 VDFSR, Virtual SError Exception Syndrome Register* on page B1-259
VDISR	RW	*B1.85 VDISR, Virtual Deferred Interrupt Status Register* on page B1-260

Virtual Memory control registers

Name	Type	Description
AMAIR0	RW	*B1.10 AMAIR0, Auxiliary Memory Attribute Indirection Register 0* on page B1-134
AMAIR1	RW	*B1.11 AMAIR1, Auxiliary Memory Attribute Indirection Register 1* on page B1-135
HAMAIR0	RW	*B1.49 HAMAIR0, Hyp Auxiliary Memory Attribute Indirection Register 0* on page B1-192
HAMAIR1	RW	*B1.50 HAMAIR1, Hyp Auxiliary Memory Attribute Indirection Register 1* on page B1-193
HTTBR	RW	*B1.55 HTTBR, Hyp Translation Table Base Register* on page B1-201
SCTLR	RW	*B1.78 SCTLR, System Control Register* on page B1-243
TTBCR	RW	*B1.80 TTBCR, Translation Table Base Control Register* on page B1-248
TTBR0	RW	*B1.82 TTBR0, Translation Table Base Register 0* on page B1-255
TTBR1	RW	*B1.83 TTBR1, Translation Table Base Register 1* on page B1-257
VTCR	RW	*B1.88 VTCR, Virtualization Translation Control Register* on page B1-265
VTTBR	RW	*B1.89 VTTBR, Virtualization Translation Table Base Register* on page B1-267

Virtualization registers

Name	Type	Description
HACR	RW	*B1.44 HACR, Hyp Auxiliary Configuration Register* on page B1-186
HACTLR	RW	*B1.45 HACTLR, Hyp Auxiliary Control Register* on page B1-187
HACTLR2	RW	*B1.46 HACTLR2, Hyp Auxiliary Control Register 2* on page B1-189
HADFSR	RW	*B1.47 HADFSR, Hyp Auxiliary Data Fault Status Syndrome Register* on page B1-190
HAISFR	RW	*B1.48 HAIFSR, Hyp Auxiliary Instruction Fault Status Syndrome Register* on page B1-191
HAMAIR0	RW	*B1.49 HAMAIR0, Hyp Auxiliary Memory Attribute Indirection Register 0* on page B1-192
HAMAIR1	RW	*B1.50 HAMAIR1, Hyp Auxiliary Memory Attribute Indirection Register 1* on page B1-193
HCR	RW	*B1.51 HCR, Hyp Configuration Register* on page B1-194
HCR2	RW	*B1.52 HCR2, Hyp Configuration Register 2* on page B1-196
HSR	RW	*B1.54 HSR, Hyp Syndrome Register* on page B1-199
HTTBR	RW	*B1.55 HTTBR, Hyp Translation Table Base Register* on page B1-201
VDFSR	RW	*B1.84 VDFSR, Virtual SError Exception Syndrome Register* on page B1-259
VDISR	RW	*B1.85 VDISR, Virtual Deferred Interrupt Status Register* on page B1-260
VMPIDR	RW	*B1.86 VMPIDR, Virtualization Multiprocessor ID Register* on page B1-263
VPIDR	RW	*B1.87 VPIDR, Virtualization Processor ID Register* on page B1-264
VTCR	RW	*B1.88 VTCR, Virtualization Translation Control Register* on page B1-265
VTTBR	RW	*B1.89 VTTBR, Virtualization Translation Table Base Register* on page B1-267

Exception and fault handling registers

Name	Type	Description
ADFSR	RW	*B1.7 ADFSR, Auxiliary Data Fault Status Register* on page B1-131
AIFSR	RW	*B1.9 AIFSR, Auxiliary Instruction Fault Status Register* on page B1-133
DFSR	RW	*B1.25 DFSR, Data Fault Status Register* on page B1-162
DISR	RW	*B1.26 DISR, Deferred Interrupt Status Register* on page B1-164
HADFSR	RW	*B1.47 HADFSR, Hyp Auxiliary Data Fault Status Syndrome Register* on page B1-190
HAIFSR	RW	*B1.48 HAIFSR, Hyp Auxiliary Instruction Fault Status Syndrome Register* on page B1-191
HSR	RW	*B1.54 HSR, Hyp Syndrome Register* on page B1-199
IFSR	RW	*B1.72 IFSR, Instruction Fault Status Register* on page B1-231
VDFSR	RW	*B1.84 VDFSR, Virtual SError Exception Syndrome Register* on page B1-259
VDISR	RW	*B1.85 VDISR, Virtual Deferred Interrupt Status Register* on page B1-260

Implementation defined registers

Name	Type	Reset	Description
CPUACTLR	RW	-	*B1.15 CPUACTLR, CPU Auxiliary Control Register* on page B1-142
CPUCFR	RO	-	*B1.16 CPUCFR, CPU Configuration Register* on page B1-144
ERXPFGCDNR	RW	-	*B1.39 ERXPFGCDNR, Selected Error Pseudo Fault Generation Count Down Register* on page B1-179
ERXPFGCTLR	RW	-	*B1.40 ERXPFGCTLR, Selected Error Pseudo Fault Generation Control Register* on page B1-181
ERXPFGFR	RO	-	*B1.41 ERXPFGFR, Selected Pseudo Fault Generation Feature Register* on page B1-183
CPUECTLR	RW	0x000000002808BC00	*B1.17 CPUECTLR, CPU Extended Control Register* on page B1-146
CPUPCR	RW	-	*B1.18 CPUPCR, CPU Private Control Register* on page B1-149
CPUPMR	RW	-	*B1.19 CPUPMR, CPU Private Mask Register* on page B1-151
CPUPOR	RW	-	*B1.20 CPUPOR, CPU Private Operation Register* on page B1-153
CPUPSELR	RW	-	*B1.21 CPUPSELR, CPU Private Selection Register* on page B1-155

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.

Table B1-5 Cluster registers (continued)

Name	Copro	CRn	Opc1	CRm	Opc2	Width	Description
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.

Legacy feature registers

Name	Type	Description
FCSEIDR	RO	*B1.43 FCSEIDR, FCSE Process ID Register* on page B1-185 In ARMv8, the core does not implement the FCSEIDR, and therefore the register is RO.

Address registers

Name	Type	Description
PAR	RW	*B1.75 PAR, Physical Address Register* on page B1-236

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.

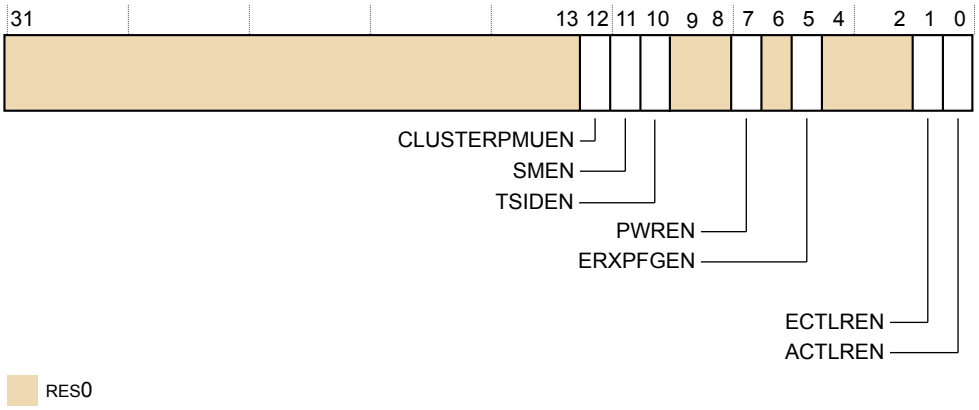


Figure B1-1 ACTLR (S) bit assignments

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

1 CPUACTLR 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 B2-289.

AArch32 register ACTLR(S) is mapped to AArch64 register ACTLR_EL3. See *B2.7 ACTLR_EL3, Auxiliary Control Register, EL3* on page B2-292.

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 ACTLR2 provides extra space to the ACTLR register to hold IMPLEMENTATION DEFINED trap functionality for execution at EL1 and EL0.

Bit field descriptions

ACTLR2 is a 32-bit register, and is part of:

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

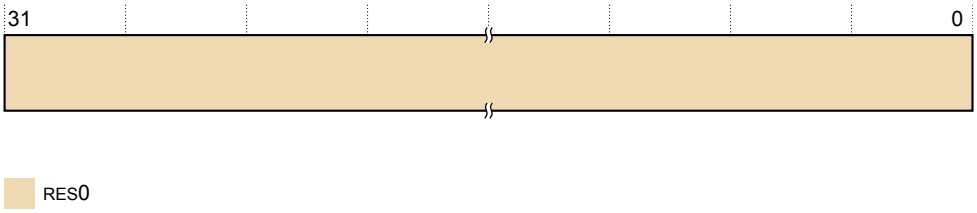


Figure B1-2 ACTLR2 bit assignments

RES0, [31:0]

RES0 Reserved.

Configurations

AArch32 System register ACTLR2 is architecturally mapped to AArch64 System register ACTLR_EL1[63:32]. See *B2.5 ACTLR_EL1, Auxiliary Control Register, EL1* on page B2-289.

AArch32 register ACTLR2(S) is architecturally mapped to AArch64 register ACTLR_EL3[63:32]. See *B2.7 ACTLR_EL3, Auxiliary Control Register, EL3* on page B2-292.

Bit fields and details not provided in this description are architecturally defined. See the *ARM® Architecture Reference Manual ARMv8, for ARMv8-A architecture profile*.

B1.7 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-A55 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.

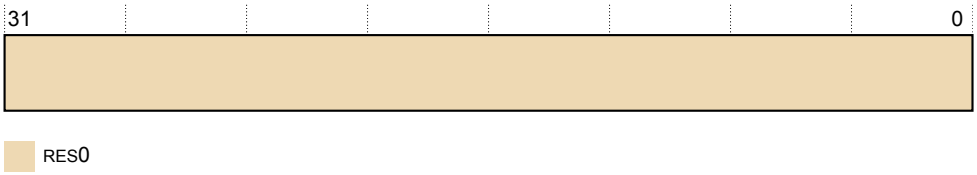


Figure B1-3 ADFSR bit assignments

[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 B2-294.

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 B2-296.

Bit fields and details not provided in this description are architecturally defined. See the *ARM® Architecture Reference Manual ARMv8, for ARMv8-A architecture profile*.

B1.8 AIDR, Auxiliary ID Register

The AIDR provides IMPLEMENTATION DEFINED identification information. This register is not used in the Cortex-A55 core.

Bit field descriptions

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

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

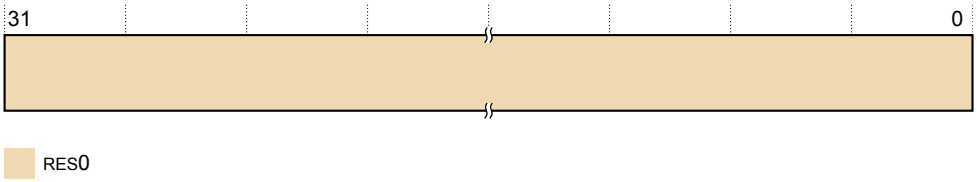


Figure B1-4 AIDR bit assignments

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 B2-300.

Bit fields and details not provided in this description are architecturally defined. See the *ARM® Architecture Reference Manual ARMv8, for ARMv8-A architecture profile*.

B1.9 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-A55 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.

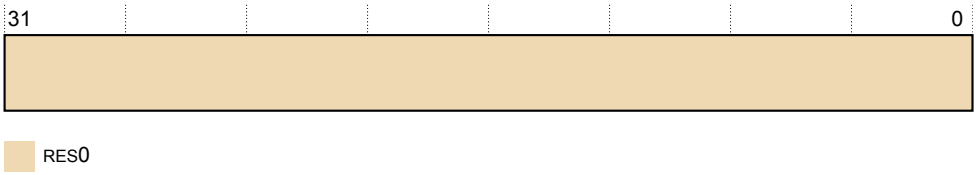


Figure B1-5 AIFSR bit assignments

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 B2-297.

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 B2-299.

Bit fields and details not provided in this description are architecturally defined. See the *ARM® Architecture Reference Manual ARMv8, for ARMv8-A architecture profile*.

B1.10 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-A55 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.

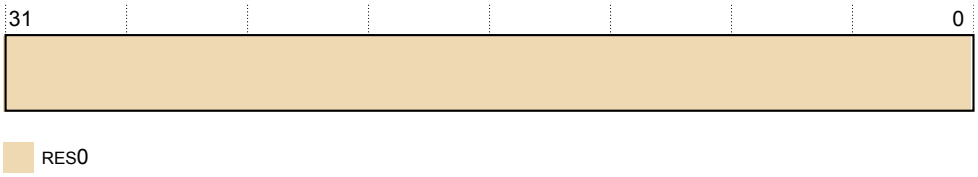


Figure B1-6 AMAIR0 bit assignments

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 B2-301.

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 B2-303.

Bit fields and details not provided in this description are architecturally defined. See the *ARM® Architecture Reference Manual ARMv8, for ARMv8-A architecture profile*.

B1.11 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-A55 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.

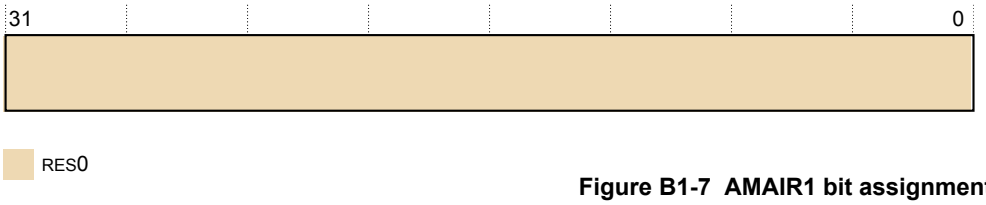


Figure B1-7 AMAIR1 bit assignments

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 B2-301.

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 B2-303.

Bit fields and details not provided in this description are architecturally defined. See the *ARM® Architecture Reference Manual ARMv8, for ARMv8-A architecture profile*.

B1.12 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.

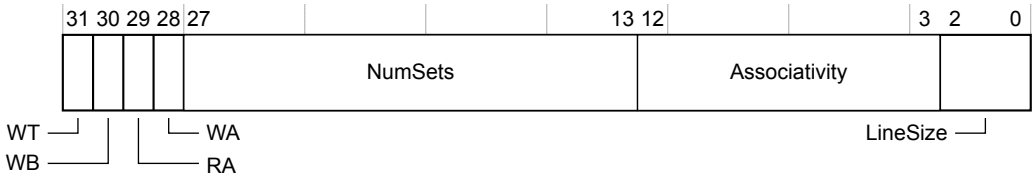


Figure B1-8 CCSIDR bit assignments

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 B1-137.

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 B1-137.

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 B1-137.

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 B1-137.

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 B1-137.

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 B1-137.

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 B1-137.

Configurations

CCSIDR is architecturally mapped to AArch64 register CCSIDR_EL1. See *B2.18 CCSIDR_EL1, Cache Size ID Register, EL1* on page B2-304.

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	16KB	7007E01A	0	1	1	1	003F	003	2
			32KB	700FE01A					007F	003	2
			64KB	701FE01A					00FF	003	2
0b000	0b1	L1 Instruction cache	16KB	2007E01A	0	0	1	0	003F	003	2
			32KB	200FE01A					007F	003	2
			64KB	201FE01A					00FF	003	2
0b001	0b0	L2 cache	Not present	See following Note.	-	-	-	-	-	-	-
			64KB	701FE01A	0	1	1	1	00FF	003	2
			128KB	703FE01A					01FF	003	2
			256KB	707FE01A					03FF	003	2
0b001	0b1	Reserved	-	-	-	-	-	-	-	-	-

Table B1-6 CCSIDR encodings (continued)

CSSELR		Cache	Size	Complete register encoding	Register bit field encoding						
Level	InD				WT	WB	RA	WA	NumSets	Associativity	LineSize
0b010	0b0	L3 cache	512KB	703FE07A	0	1	1	1	01FF	00F	2
			1024KB	707FE07A					03FF	00F	2
			2048KB	70FFE07A					07FF	00F	2
			4096KB	71FFE07A					0FFF	00F	2
0b0101 - 0b1111		Reserved	-	-	-	-	-	-	-	-	-

Note

If no L2 cache is present the core uses L3 cache as L2, and the L3 encodings apply.

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.

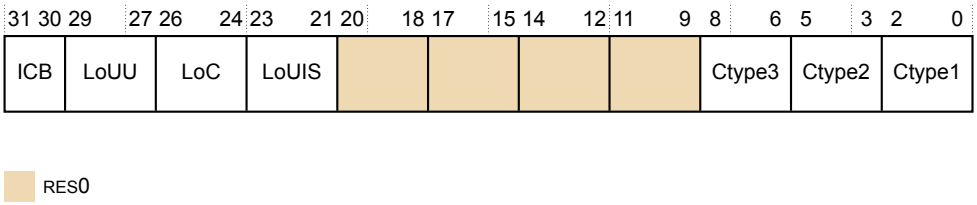


Figure B1-9 CLIDR bit assignments

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:

`0b0000` 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` L2 and L3 cache are implemented.

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

`0b000` No levels of cache need to cleaned or invalidated when cleaning or invalidating to the 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 cluster implements L3 cache. If present, unified instruction and data caches at Level-3:

`0b000` L3 cache is not implemented.

`0b100` L3 cache is implemented.

If Ctype2 has a value of `0b000`, the value of Ctype3 must be ignored.

Ctype2, [5:3]

Indicates the type of cache if the core implements L2 cache. If present, 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 B2-306.

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.

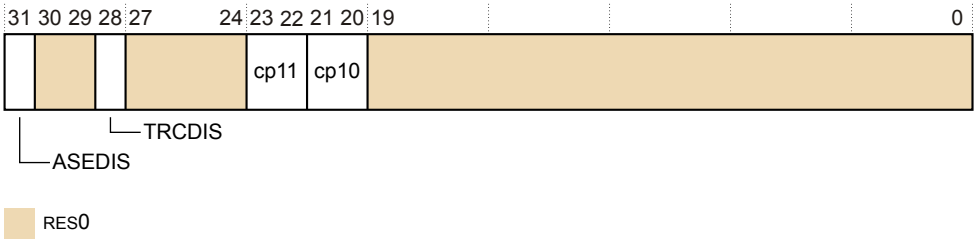


Figure B1-10 CPACR bit assignments

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 B2-308.

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.

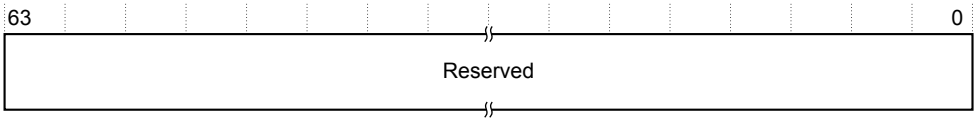


Figure B1-11 CPUACTLR bit assignments

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 B2-311.

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 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.

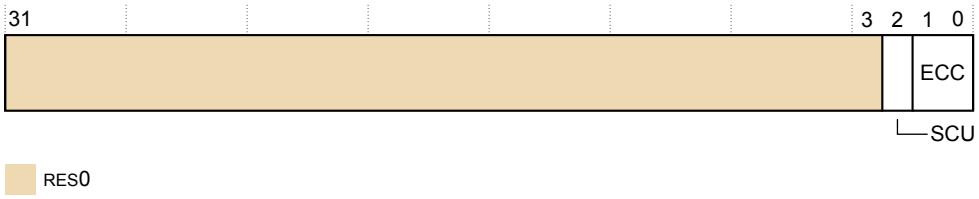


Figure B1-12 CPUCFR bit assignments

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 *B2.24 CPUCFR_EL1, CPU Configuration Register, EL1* on page B2-313.

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.

B1.17 CPUECTLR, CPU Extended Control Register

The CPUECTLR provides extra 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.

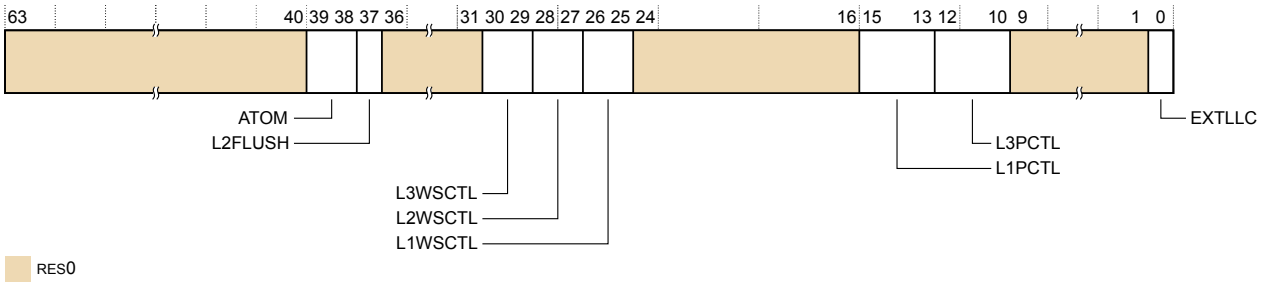


Figure B1-13 CPUECTLR bit assignments

RES0, [63:40]

RES0 Reserved.

ATOM, [39:38]

00 Atomic instructions are performed near if they hit in the cache in a unique state, or far if they miss or are shared. For more details, see *A6.4.1 Memory system implementation* on page A6-78. This is the default.

01 Force all cacheable atomic instructions to be executed near, in the L1 cache.

10 Force most cacheable atomic instructions to be executed far, in the L3 cache or beyond.

11 Force cacheable load atomics, including SWP and CAS, to be executed near, in the L1 cache. Store atomics are performed near if they hit in the cache in a unique state, or far if they miss or are shared.

L2FLUSH, [37]

0 L2 cache flushes, for example during a core powerdown sequence, cause clean lines to be allocated into the L3 cache rather than discarding them. This can improve performance if it is known that the data is likely to be used soon by another core.

RES0, [36:31]

RES0 Reserved.

L3WSCTL, [30:29]

Write streaming no-L3-allocate threshold. The possible values are:

00 128th consecutive streaming cache line does not allocate in the L1, L2, or L3 cache.

01 1024th consecutive streaming cache line does not allocate in the L1, L2, or L3 cache. This is the reset value.

10 4096th consecutive streaming cache line does not allocate in the L1, L2, or L3 cache.

11 Disables streaming. All write-allocate lines allocate in the L1, L2, or L3 cache.

L2WSCTL, [28:27]

Write streaming no-L2-allocate threshold. The possible values are:

00 16th consecutive streaming cache line does not allocate in the L1 or L2 cache.

01 128th consecutive streaming cache line does not allocate in the L1 or L2 cache. This is the reset value.

10 512th consecutive streaming cache line does not allocate in the L1 or L2 cache.

11 Disables streaming. All write-allocate lines allocate in the L1 or L2 cache.

L1WSCTL, [26:25]

Write streaming no-L1-allocate threshold. The possible values are:

00 4th consecutive streaming cache line does not allocate in the L1 cache. This is the reset value.

01 64th consecutive streaming cache line does not allocate in the L1 cache.

10 128th consecutive streaming cache line does not allocate in the L1

11 Disables streaming. All write-allocate lines allocate in the L1 cache.

RES0, [24:16]

RES0 Reserved.

L1PCTL, [15:13]

L1 Data prefetch control. The value of the L1PCTL field determines the maximum number of outstanding data prefetches allowed in the L1 memory system (not counting the data prefetches generated by software load/PLD instructions).

000 Prefetch disabled.

001 1 outstanding prefetch allowed.

010 2 outstanding prefetches allowed.

011 3 outstanding prefetches allowed.

100 4 outstanding prefetches allowed.

101 5 outstanding prefetches allowed. This is the reset value.

110 6 outstanding prefetches allowed.

111 7 outstanding prefetches allowed.

L3PCTL, [12:10]

L3 Data prefetch control. The value of the L3PCTL field determines the approximate distance between the L1 prefetcher and requests sent to the L3 memory system. Increasing this distance may improve performance on systems with higher latency to main memory, but increasing it too far can reduce performance.

——— **Note** ———

The L3 memory system may have more outstanding access to the system than this number.

———————————

000 Fetch 16 lines ahead.

001 Fetch 32 lines ahead.

010 Reserved.

011 Reserved.

100 Disable L3 prefetching.

101 Fetch 2 lines ahead.

110 Fetch 4 lines ahead.

111 Fetch 8 lines ahead. This is the reset value.

RES0, [9:1]

RES0 Reserved.

EXTLLC, [0]

`0` 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 *B2.25 CPUECTLR_EL1, CPU Extended Control Register, EL1* on page B2-315.

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 MRRC 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, 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 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.

B1.18 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.

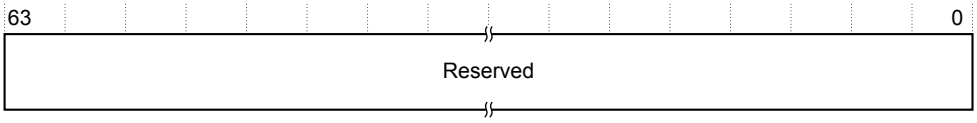


Figure B1-14 CPUPCR bit assignments

Reserved, [63:0]

Reserved for ARM internal use.

Configurations

CPUPCR is:

- Only accessible in Secure state.
- Mapped to the AArch64 CPUPCR_EL3 register. See *B2.26 CPUPCR_EL3, CPU Private Control Register, EL3* on page B2-318.

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.

B1.19 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.

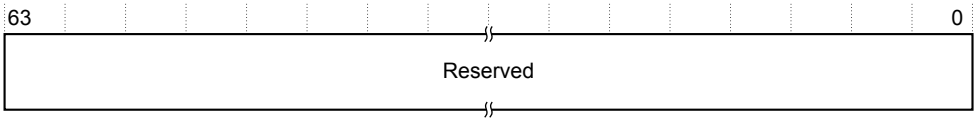


Figure B1-15 CPUPMR bit assignments

Reserved, [63:0]

Reserved for ARM internal use.

Configurations

CPUPMR is:

- Only accessible in Secure state.
- Mapped to the AArch64 CPUPMR_EL3 register. See *B2.27 CPUPMR_EL3, CPU Private Mask Register, EL3* on page B2-320.

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.

B1.20 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.

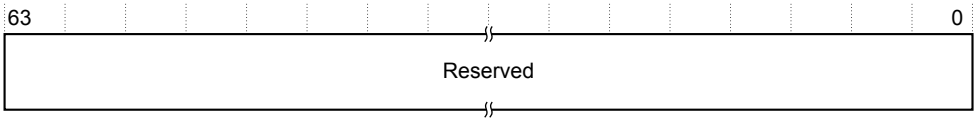


Figure B1-16 CPUPOR bit assignments

Reserved, [63:0]

Reserved for ARM internal use.

Configurations

CPUPOR is:

- Only accessible in Secure state.
- Mapped to the AArch64 CPUPOR_EL3 register. See *B2.28 CPUPOR_EL3, CPU Private Operation Register, EL3* on page B2-322.

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.

B1.21 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.

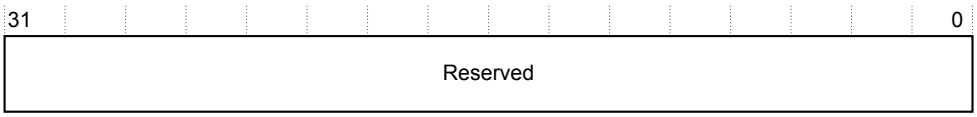


Figure B1-17 CPUPSELR bit assignments

Reserved, [31:0]

Reserved for ARM internal use.

Configurations

CPUPSELR is:

- Only accessible in Secure state.
- Mapped to the AArch64 CPUPSELR_EL3 register. See *B2.29 CPUPSELR_EL3, CPU Private Selection Register, EL3* on page B2-324.

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.

B1.22 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.

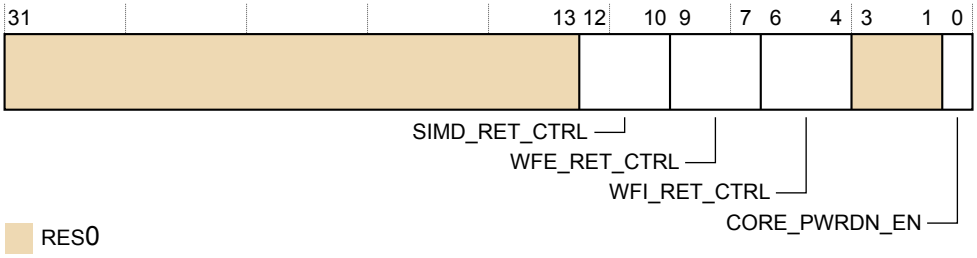


Figure B1-18 CPUPWRCTLR bit assignments

RES0, [31:13]

RES0 Reserved.

SIMD_RET_CTRL, [12:10]

Advanced SIMD and floating-point retention control:

0b000 Disable the retention circuit. This is the default value, see *Table B2-7 CPUPWRCTLR Retention Control Field* on page B2-327 for more retention control options.

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 B1-157 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 B1-157 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

Table B1-7 CPUPWRCTLR Retention Control Field (continued)

Encoding	Note	Minimum Delay before retention 50MHz – 10MHz
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
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 *B2.30 CPUPWRCTLR_EL1, Power Control Register, EL1* on page B2-326.

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.

B1.23 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 B1-136.

Bit field descriptions

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

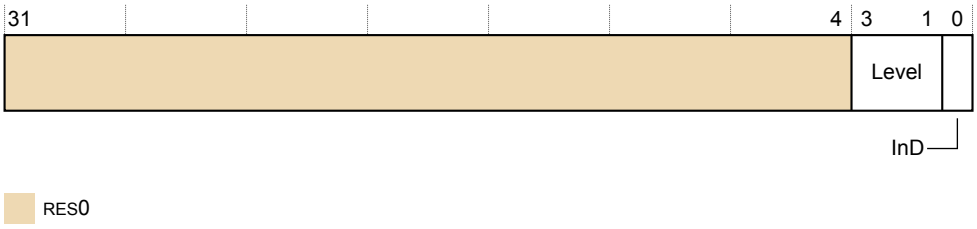


Figure B1-19 CSSELR bit assignments

RES0, [31:4]

RES0 Reserved.

Level, [3:1]

Cache level of required cache:

`0b000` L1.

`0b001` L2.

Only if L2 is present, or if no L2 present then L3 is present.

`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 *B2.31 CSSELR_EL1, Cache Size Selection Register, EL1* on page B2-328.

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*.

B1.24 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.

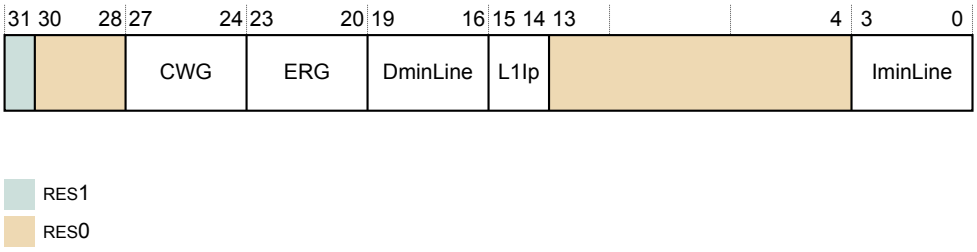


Figure B1-20 CTR bit assignments

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 *B2.32 CTR_EL0, Cache Type Register, EL0* on page B2-329.

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*.

B1.25 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.

- *B1.25.1 DFSR with Short-descriptor translation table format* on page B1-162.
- *B1.25.2 DFSR with Long-descriptor translation table format* on page B1-162.

Configurations

DFSR (NS) is architecturally mapped to AArch64 register ESR_EL1. See *B2.46 ESR_EL1, Exception Syndrome Register, EL1* on page B2-347.

DFSR (S) is architecturally mapped to AArch64 register ESR_EL3. See *B2.48 ESR_EL3, Exception Syndrome Register, EL3* on page B2-349.

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:

- *B1.25.1 DFSR with Short-descriptor translation table format* on page B1-162.
- *B1.25.2 DFSR with Long-descriptor translation table format* on page B1-162.

B1.25.1 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:

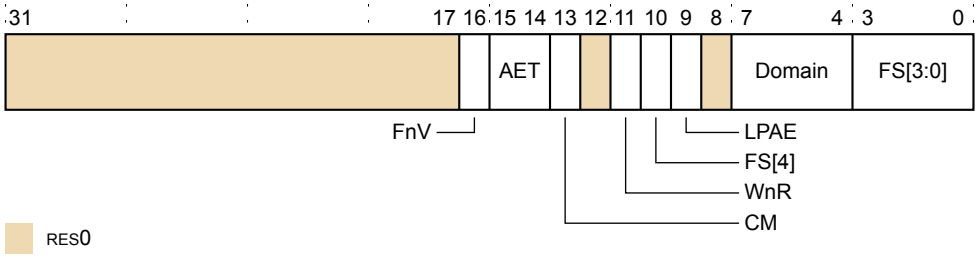


Figure B1-21 DFSR bit assignments for Short-descriptor translation table format

AET, [15:14]

Asynchronous Error Type. Describes the state of the PE after taking an asynchronous Data Abort exception. The value is:

`0b01` *Uncorrected error, Unrecoverable error* (UEU).

ExT, [12]

RES0 Reserved. This bit is unused.

B1.25.2 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:

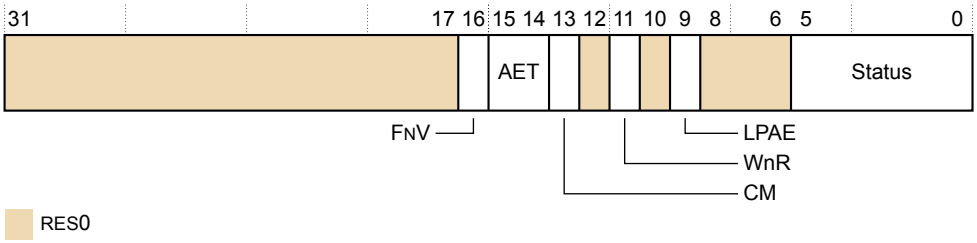


Figure B1-22 DFSR bit assignments for Long-descriptor translation table format

AET, [15:14]

Asynchronous Error Type. Describes the state of the PE after taking an asynchronous Data Abort exception. The value is:

`0b01` *Uncorrected error, Unrecoverable error* (UEU).

ExT, [12]

RES0 Reserved. This bit is unused.

B1.26 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 *B1.26.1 DISR with Short-descriptor translation table format* on page B1-164.
- When written at EL1 using Long-descriptor format. See *B1.26.2 DISR with Long-descriptor translation table format* on page B1-165.
- When written at EL2. See *B1.26.3 DISR at EL2* on page B1-166.

Configurations

AArch32 register DISR is architecturally mapped to AArch64 register DISR_EL1. See *B2.34 DISR_EL1, Deferred Interrupt Status Register, EL1* on page B2-332.

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:

B1.26.1 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:

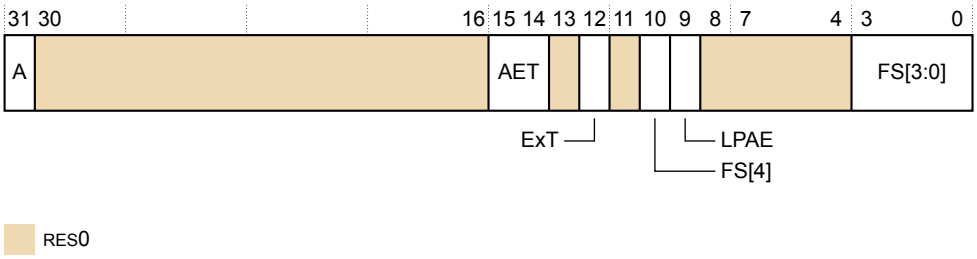


Figure B1-23 DISR bit assignments for Short-descriptor translation table format

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 PE after taking an asynchronous Data Abort exception. The value is:

`0b01` *Uncorrected error, Unrecoverable error* (UEU).

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.

B1.26.2 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:

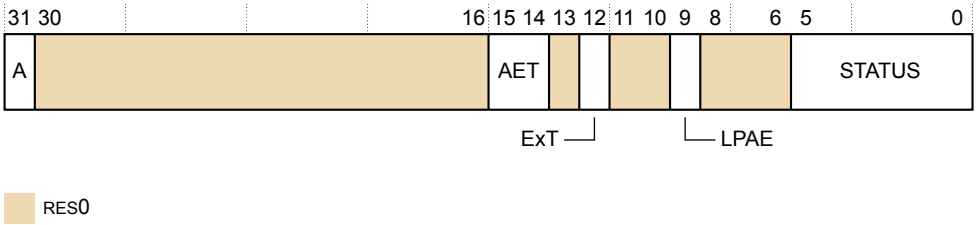


Figure B1-24 DISR bit assignments for Long-descriptor translation table format

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:

`0b01` *Uncorrected error, Unrecoverable error* (UEU).

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.

B1.26.3 DISR at EL2

DISR has a specific format when written at EL2.

The following figure shows the DISR bit assignments when written at EL2:

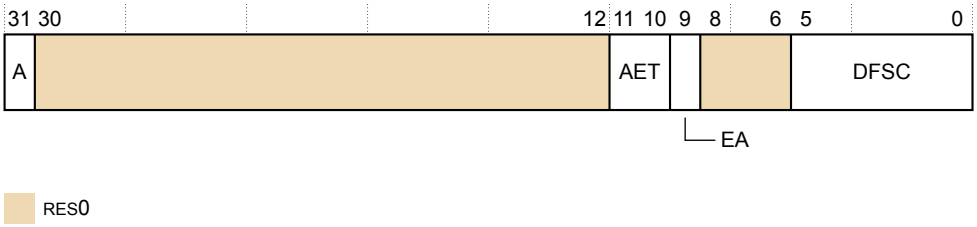


Figure B1-25 DISR bit assignments for Long-descriptor translation table format

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:

0b01 *Uncorrected error, Unrecoverable error* (UEU).

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.

B1.27 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.

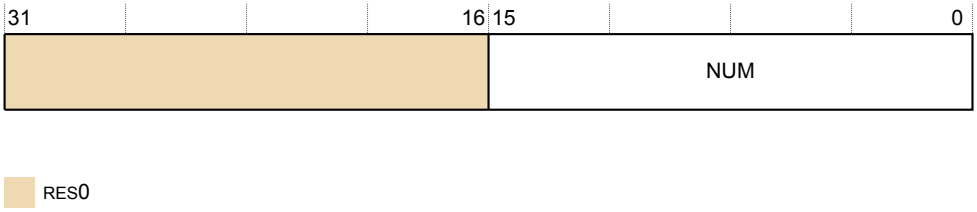


Figure B1-26 ERRIDR bit assignments

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 *B2.35 ERRIDR_EL1, Error ID Register, EL1* on page B2-334.

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*.

B1.28 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.

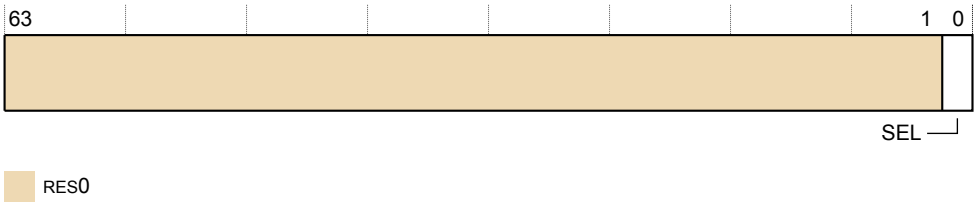


Figure B1-27 ERRSELR bit assignments

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-A55 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 *B2.36 ERRSELR_EL1, Error Record Select Register, EL1* 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*.

B1.29 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 *ERR0ADDR, Error Record Address Register*.

If ERRSELR.SEL==1, then ERXADDR accesses the ERR1ADDR register of the DSU error record. See the *ARM® DynamIQ™ Shared Unit Technical Reference Manual*.

B1.30 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 B3-436.

If ERRSELR.SEL==1, then ERXADDR2 accesses the ERR1ADDR register of the DSU error record. See the *ARM® DynamIQ™ Shared Unit Technical Reference Manual*.

B1.31 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 *ERR0CTLR, Error Record Control Register*.

If ERRSELR.SEL==1, then ERXCLTR accesses the ERR1CTLR register of the DSU error record. See the *ARM® DynamIQ™ Shared Unit Technical Reference Manual*.

B1.32 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 *ERR0CTLR, Error Record Control Register*.

If ERRSELR.SEL==1, then ERXCLTR2 accesses the ERR1CTLR register of the DSU error record. See the *ARM® DynamIQ™ Shared Unit Technical Reference Manual*.

B1.33 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 B3-439.

If ERRSELR.SEL==1, then ERXCLTR accesses the ERR1FR register of the DSU error record. See the *ARM® DynamIQ™ Shared Unit Technical Reference Manual*.

B1.34 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 B3-439.

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 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 B3-441.

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*.

B1.36 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 B3-441.

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*.

B1.37 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 B3-443.

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*.

B1.38 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 B3-443.

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*.

B1.39 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 B3-444.

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 *B2.42 ERXPFGCDNR_EL1, Selected Error Pseudo Fault Generation Count Down Register, EL1* on page B2-341.

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 B1-128 and *B1.45 HACTLR, Hyp Auxiliary Control Register* on page B1-187.

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 B2-290 and *B2.7 ACTLR_EL3, Auxiliary Control Register, EL3* on page B2-292.

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.

B1.40 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 B3-445.

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 *B2.43 ERXPFGCTLR_EL1, Selected Error Pseudo Fault Generation Control Register, EL1* on page B2-343.

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 B1-128 and *B1.45 HACTLR, Hyp Auxiliary Control Register* on page B1-187.

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 B2-290 and *B2.7 ACTLR_EL3, Auxiliary Control Register, EL3* on page B2-292.

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.

B1.41 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 B3-447.

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 *B2.44 ERXPFGFR_EL1, Selected Pseudo Fault Generation Feature Register, EL1* on page B2-345.

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.

B1.42 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 B3-449.

If ERRSELR.SEL==1, then ERXSTATUS accesses the ERR1STATUS register of the DSU error record. See the *ARM® DynamIQ™ Shared Unit Technical Reference Manual*.

B1.43 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.

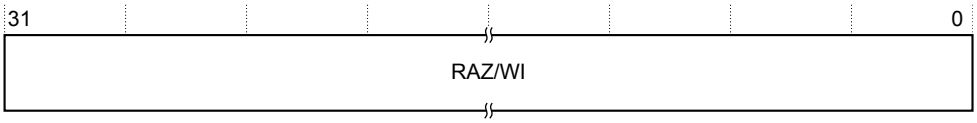


Figure B1-28 FCSEIDR bit assignments

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*.

B1.44 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-A55 core.

Bit field descriptions

HACR is a 32-bit register, and is part of the Virtualization registers functional group.

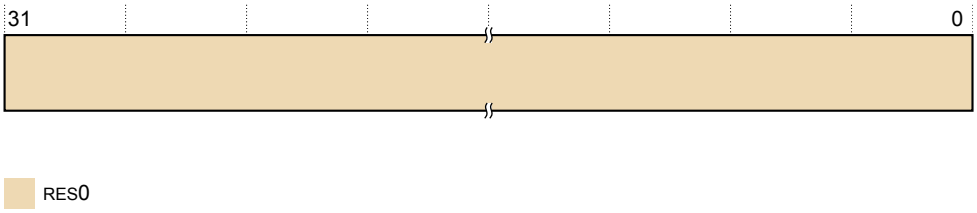


Figure B1-29 HACR bit assignments

RES0, [31:0]

RES0 Reserved.

Configurations

AArch32 System register HACR is architecturally mapped to AArch64 System register HACR_EL2. See *B2.49 HACR_EL2, Hyp Auxiliary Configuration Register, EL2* on page B2-350.

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 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.

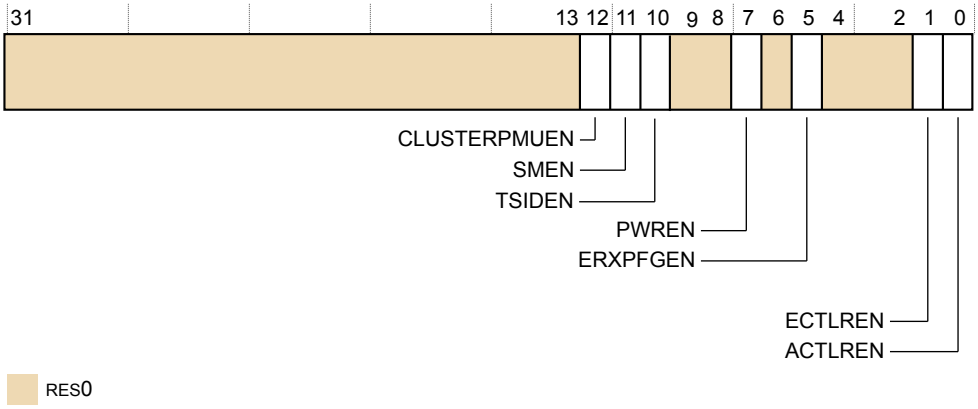


Figure B1-30 HACTLR bit assignments

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 and CLUSTERACTLR are not write accessible from EL1. This is the reset value.

1 CPUACTLR 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 B2-290.

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 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.

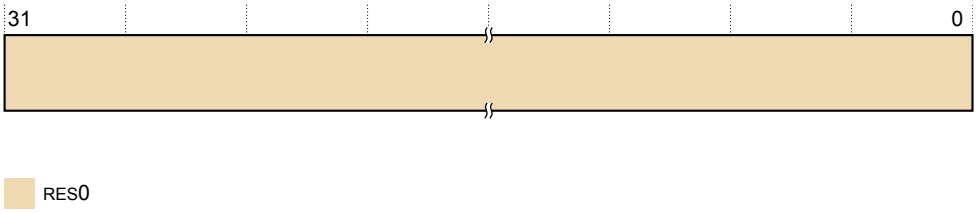


Figure B1-31 HACTLR2 bit assignments

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 B2-290.

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*.

B1.47 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-A55 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.

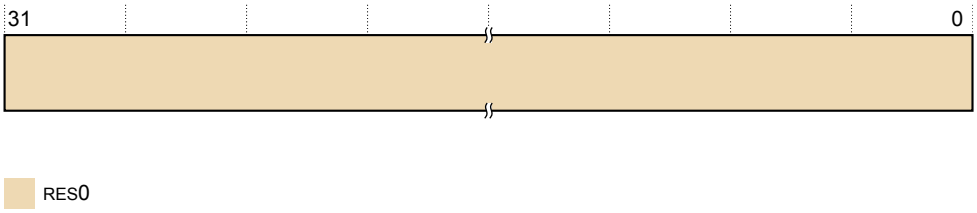


Figure B1-32 HADFSR bit assignments

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 B2-295.

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 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-A55 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.

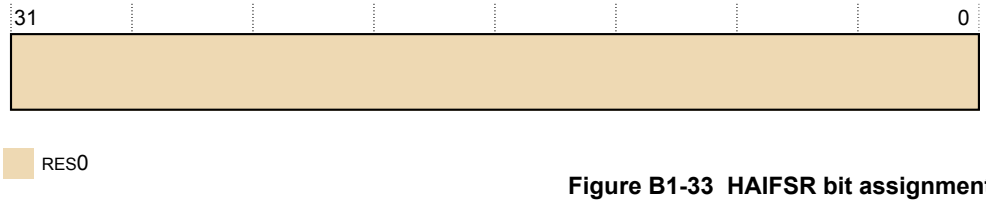


Figure B1-33 HAIFSR bit assignments

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 B2-298.

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 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-A55 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.

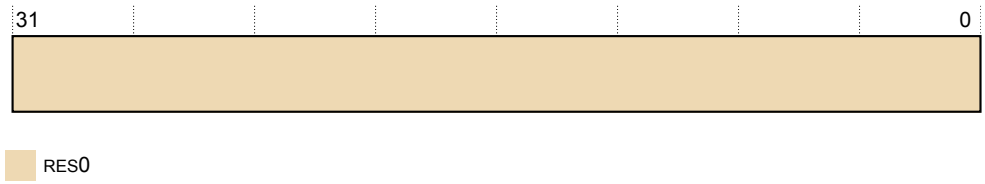


Figure B1-34 HAMAIR0 bit assignments

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 B2-302.

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 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-A55 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.

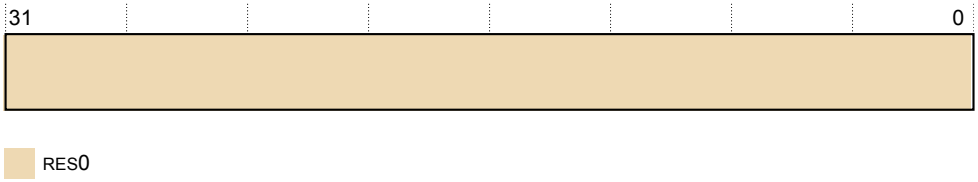


Figure B1-35 HAMAIR1 bit assignments

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 B2-302.

Bit fields and details not provided in this description are architecturally defined. See the *ARM® Architecture Reference Manual ARMv8, for ARMv8-A architecture profile*.

B1.51 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`.

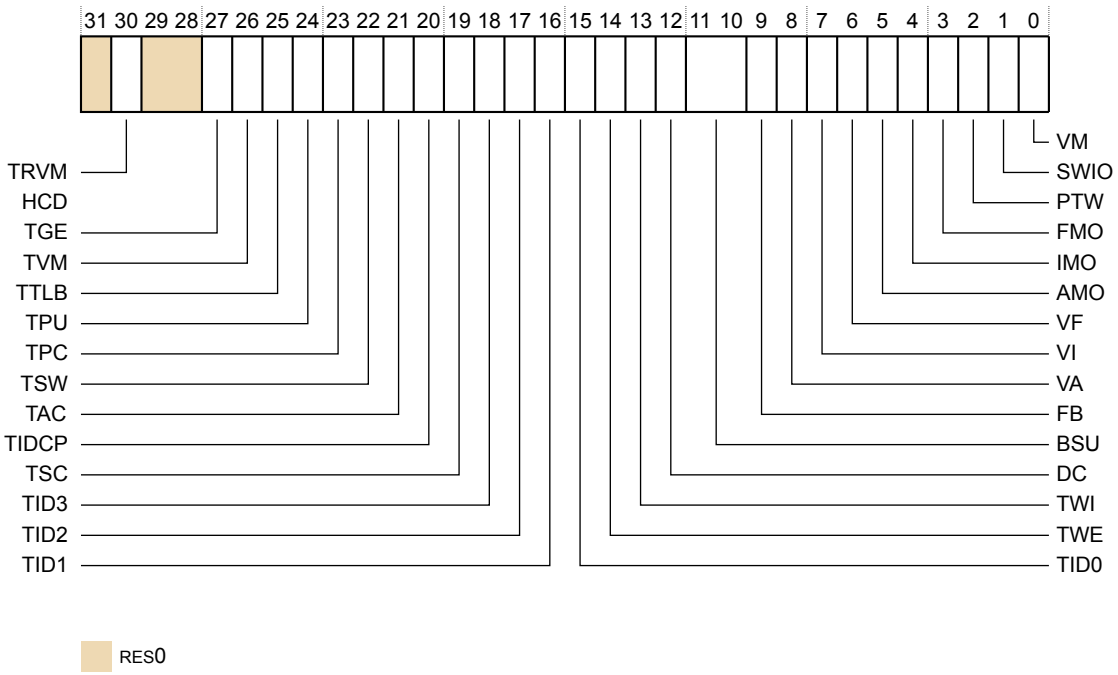


Figure B1-36 HCR bit assignments

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-A55 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 *B2.50 HCR_EL2, Hypervisor Configuration Register, EL2* on page B2-351.

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 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`.

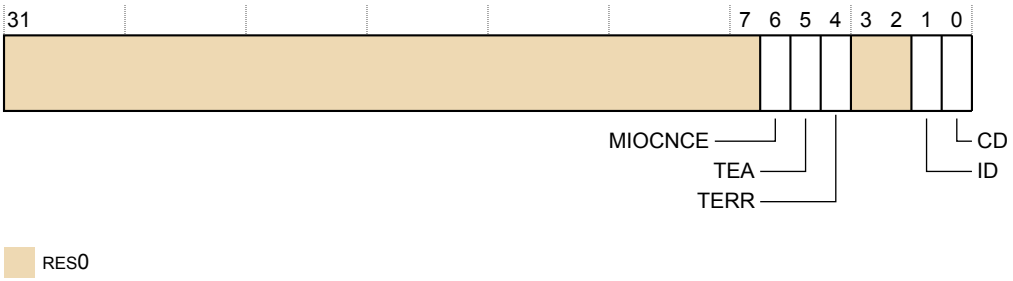


Figure B1-37 HCR2 bit assignments

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 *B2.50 HCR_EL2, Hypervisor Configuration Register, EL2* on page B2-351.

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*.

B1.53 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.

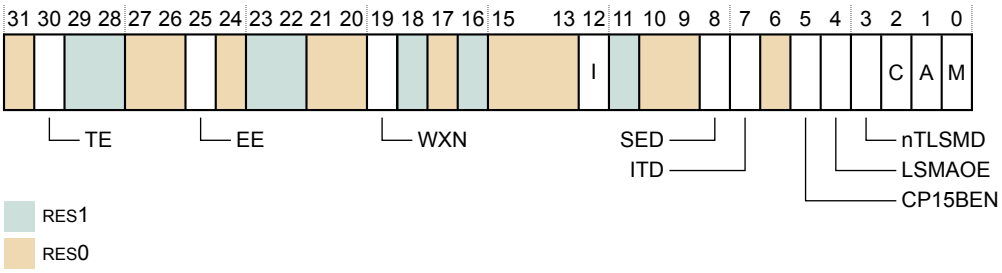


Figure B1-38 HSCTLR bit assignments

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 *B2.88 SCTLR_EL2, System Control Register, EL2* on page B2-410.

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 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.

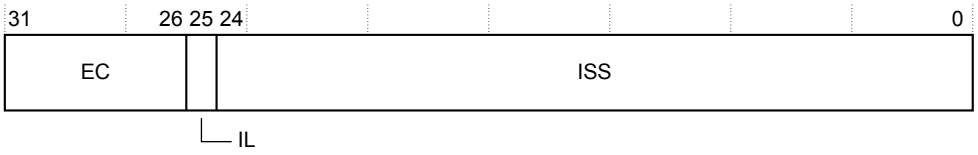


Figure B1-39 HSR bit assignments

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 *B1.54.1 Encoding of ISS[24:20] when HSR[31:30] is 0b00* on page B1-199.

Configurations
HSR is architecturally mapped to AArch64 register ESR_EL2. See *B2.47 ESR_EL2, Exception Syndrome Register, EL2* on page B2-348.

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:

B1.54.1 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.

B1.55 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.

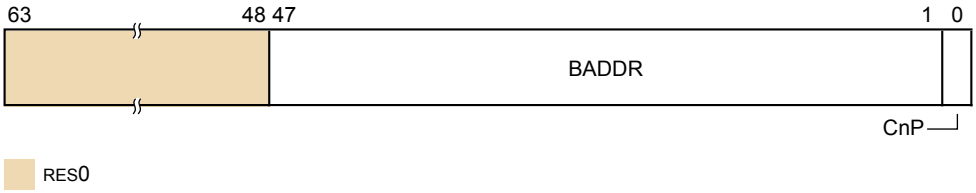


Figure B1-40 HTTBR bit assignments

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*.

B1.56 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-A55 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.

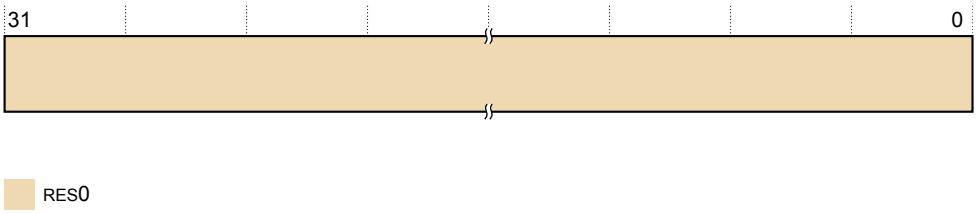


Figure B1-41 ID_AFR0 bit assignments

RES0, [31:0]

Reserved, RES0.

Configurations

AArch32 System register ID_AFR0 is architecturally mapped to AArch64 System register ID_AFR0_EL1. See *B2.59 ID_AFR0_EL1, AArch32 Auxiliary Feature Register 0, EL1* on page B2-365.

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_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.

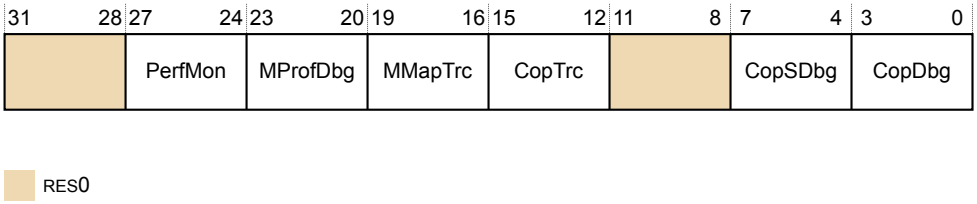


Figure B1-42 ID_DFR0 bit assignments

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 *B2.60 ID_DFR0_EL1, AArch32 Debug Feature Register 0, EL1* on page B2-366.

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*.

B1.58 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.

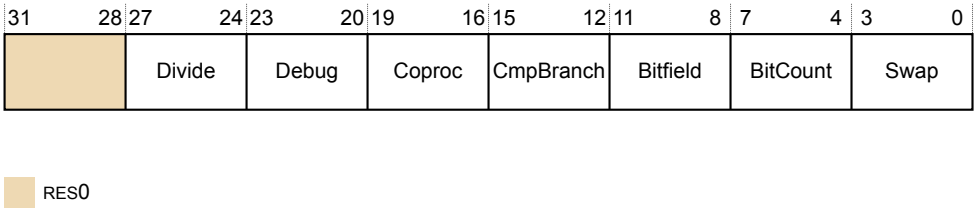


Figure B1-43 ID_ISAR0 bit assignments

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 *B2.61 ID_ISAR0_EL1, AArch32 Instruction Set Attribute Register 0, EL1* on page B2-368.

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:

- *B1.59 ID_ISAR1, Instruction Set Attribute Register 1* on page B1-207.
- *B1.60 ID_ISAR2, Instruction Set Attribute Register 2* on page B1-209.
- *B1.61 ID_ISAR3, Instruction Set Attribute Register 3* on page B1-211.
- *B1.62 ID_ISAR4, Instruction Set Attribute Register 4* on page B1-213.
- *B1.63 ID_ISAR5, Instruction Set Attribute Register 5* on page B1-215.

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_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.

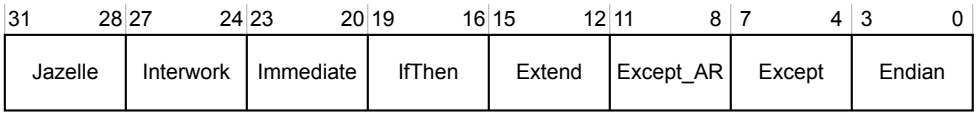


Figure B1-44 ID_ISAR1 bit assignments

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 *B2.62 ID_ISAR1_EL1, AArch32 Instruction Set Attribute Register 1, EL1* on page B2-370.

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:

- *B1.58 ID_ISAR0, Instruction Set Attribute Register 0* on page B1-205.
- *B1.60 ID_ISAR2, Instruction Set Attribute Register 2* on page B1-209.
- *B1.61 ID_ISAR3, Instruction Set Attribute Register 3* on page B1-211.
- *B1.62 ID_ISAR4, Instruction Set Attribute Register 4* on page B1-213.
- *B1.63 ID_ISAR5, Instruction Set Attribute Register 5* on page B1-215.

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_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.

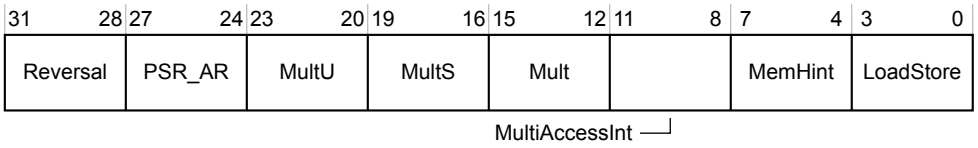


Figure B1-45 ID_ISAR2 bit assignments

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 *B2.63 ID_ISAR2_EL1, AArch32 Instruction Set Attribute Register 2, EL1* on page B2-372.

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:
- *B1.58 ID_ISAR0, Instruction Set Attribute Register 0* on page B1-205.
- *B1.59 ID_ISAR1, Instruction Set Attribute Register 1* on page B1-207.
- *B1.61 ID_ISAR3, Instruction Set Attribute Register 3* on page B1-211.
- *B1.62 ID_ISAR4, Instruction Set Attribute Register 4* on page B1-213.
- *B1.63 ID_ISAR5, Instruction Set Attribute Register 5* on page B1-215.

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_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.

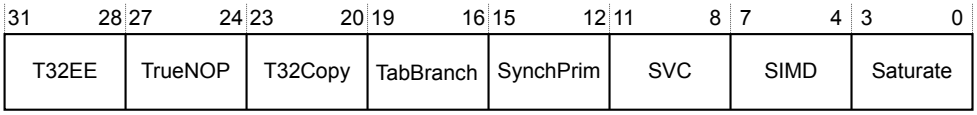


Figure B1-46 ID_ISAR3 bit assignments

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 *B2.64 ID_ISAR3_EL1, AArch32 Instruction Set Attribute Register 3, EL1* on page B2-374.

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:
- *B1.58 ID_ISAR0, Instruction Set Attribute Register 0* on page B1-205.
- *B1.59 ID_ISAR1, Instruction Set Attribute Register 1* on page B1-207.
- *B1.60 ID_ISAR2, Instruction Set Attribute Register 2* on page B1-209.
- *B1.62 ID_ISAR4, Instruction Set Attribute Register 4* on page B1-213.
- *B1.63 ID_ISAR5, Instruction Set Attribute Register 5* on page B1-215.

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_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.

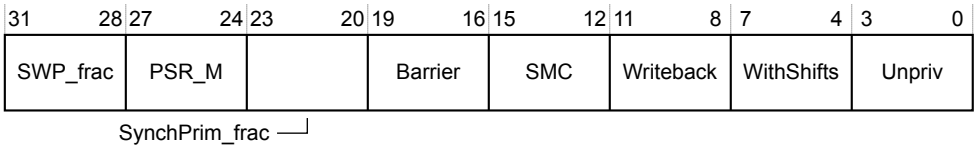


Figure B1-47 ID_ISAR4 bit assignments

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 *B2.65 ID_ISAR4_EL1, AArch32 Instruction Set Attribute Register 4, 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_ISAR0, ID_ISAR1, ID_ISAR2, ID_ISAR3, and ID_ISAR5. See:

- *B1.58 ID_ISAR0, Instruction Set Attribute Register 0* on page B1-205.
- *B1.59 ID_ISAR1, Instruction Set Attribute Register 1* on page B1-207.
- *B1.60 ID_ISAR2, Instruction Set Attribute Register 2* on page B1-209.
- *B1.61 ID_ISAR3, Instruction Set Attribute Register 3* on page B1-211.
- *B1.63 ID_ISAR5, Instruction Set Attribute Register 5* on page B1-215.

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_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.

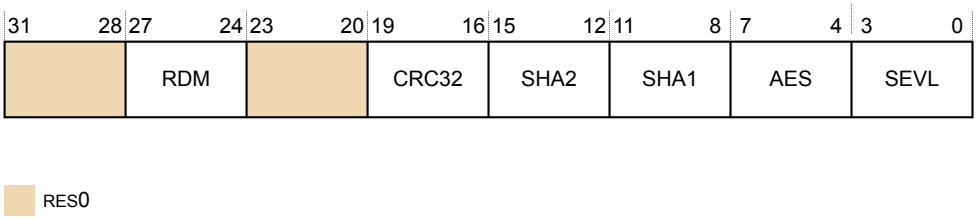


Figure B1-48 ID_ISAR5 bit assignments

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 *B2.66 ID_ISAR5_EL1, AArch32 Instruction Set Attribute Register 5, EL1* on page B2-378.

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:

- *B1.58 ID_ISAR0, Instruction Set Attribute Register 0* on page B1-205.
- *B1.59 ID_ISAR1, Instruction Set Attribute Register 1* on page B1-207.
- *B1.60 ID_ISAR2, Instruction Set Attribute Register 2* on page B1-209.
- *B1.61 ID_ISAR3, Instruction Set Attribute Register 3* on page B1-211.
- *B1.62 ID_ISAR4, Instruction Set Attribute Register 4* on page B1-213.

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_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.

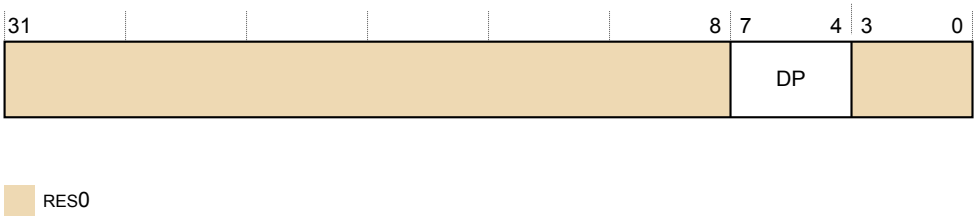


Figure B1-49 ID_ISAR6 bit assignments

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 *B2.67 ID_ISAR6_EL1, AArch32 Instruction Set Attribute Register 6, EL1* on page B2-380.

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:

- *B1.58 ID_ISAR0, Instruction Set Attribute Register 0* on page B1-205.
- *B1.59 ID_ISAR1, Instruction Set Attribute Register 1* on page B1-207.
- *B1.60 ID_ISAR2, Instruction Set Attribute Register 2* on page B1-209.
- *B1.61 ID_ISAR3, Instruction Set Attribute Register 3* on page B1-211.
- *B1.62 ID_ISAR4, Instruction Set Attribute Register 4* on page B1-213.
- *B1.63 ID_ISAR5, Instruction Set Attribute Register 5* on page B1-215.

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_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.

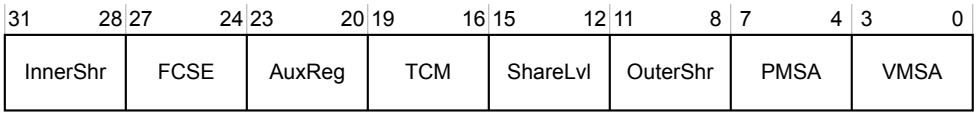


Figure B1-50 ID_MMFR0 bit assignments

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 *B2.68 ID_MMFR0_EL1, AArch32 Memory Model Feature Register 0, EL1* on page B2-381.

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:

- *B1.66 ID_MMFR1, Memory Model Feature Register 1* on page B1-220.
- *B1.67 ID_MMFR2, Memory Model Feature Register 2* on page B1-222.
- *B1.68 ID_MMFR3, Memory Model Feature Register 3* on page B1-224.
- *B1.69 ID_MMFR4, Memory Model Feature Register 4* on page B1-226.

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_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.

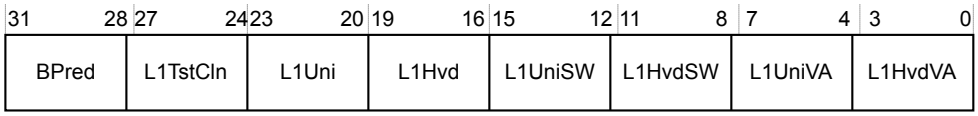


Figure B1-51 ID_MMFR1 bit assignments

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 *B2.69 ID_MMFR1_EL1, AArch32 Memory Model Feature Register 1, EL1* on page B2-383.

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:

- *B1.65 ID_MMFR0, Memory Model Feature Register 0* on page B1-218.
- *B1.67 ID_MMFR2, Memory Model Feature Register 2* on page B1-222.
- *B1.68 ID_MMFR3, Memory Model Feature Register 3* on page B1-224.
- *B1.69 ID_MMFR4, Memory Model Feature Register 4* on page B1-226.

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_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.

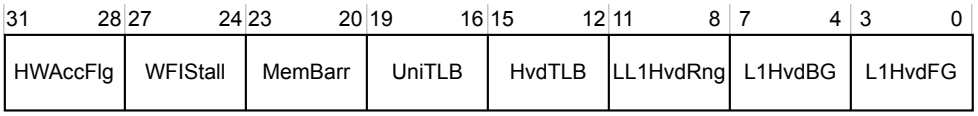


Figure B1-52 ID_MMFR2 bit assignments

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 *B2.70 ID_MMFR2_EL1, AArch32 Memory Model Feature Register 2, EL1* on page B2-385.

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:

- *B1.65 ID_MMFR0, Memory Model Feature Register 0* on page B1-218.
- *B1.66 ID_MMFR1, Memory Model Feature Register 1* on page B1-220.
- *B1.68 ID_MMFR3, Memory Model Feature Register 3* on page B1-224.
- *B1.69 ID_MMFR4, Memory Model Feature Register 4* on page B1-226.

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_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.

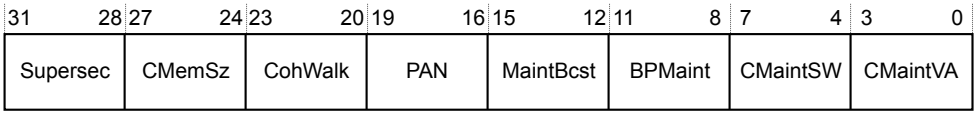


Figure B1-53 ID_MMFR3 bit assignments

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 *B2.71 ID_MMFR3_EL1, AArch32 Memory Model Feature Register 3, EL1* on page B2-387.

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:
- *B1.65 ID_MMFR0, Memory Model Feature Register 0* on page B1-218.
- *B1.66 ID_MMFR1, Memory Model Feature Register 1* on page B1-220.
- *B1.67 ID_MMFR2, Memory Model Feature Register 2* on page B1-222.
- *B1.69 ID_MMFR4, Memory Model Feature Register 4* on page B1-226.

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_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.

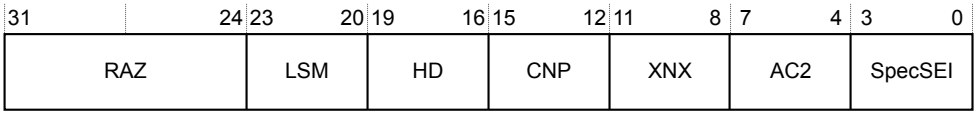


Figure B1-54 ID_MMFR4 bit assignments

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 *B2.72 ID_MMFR4_EL1, AArch32 Memory Model Feature Register 4, 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_MMFR0, ID_MMFR1, ID_MMFR2, and ID_MMFR3. See:

- *B1.65 ID_MMFR0, Memory Model Feature Register 0* on page B1-218.
- *B1.66 ID_MMFR1, Memory Model Feature Register 1* on page B1-220.
- *B1.67 ID_MMFR2, Memory Model Feature Register 2* on page B1-222.
- *B1.68 ID_MMFR3, Memory Model Feature Register 3* 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*.

B1.70 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.

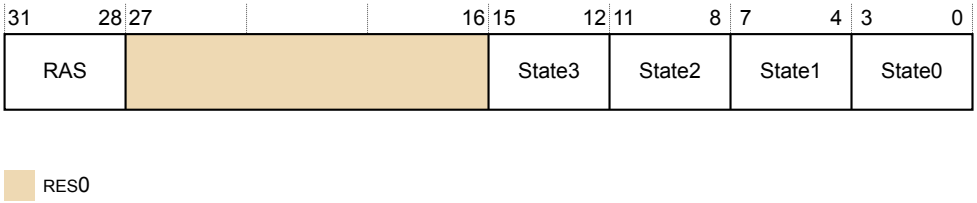


Figure B1-55 ID_PFR0 bit assignments

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 *B2.73 ID_PFR0_EL1, AArch32 Processor Feature Register 0, EL1* on page B2-391.

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*.

B1.71 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.

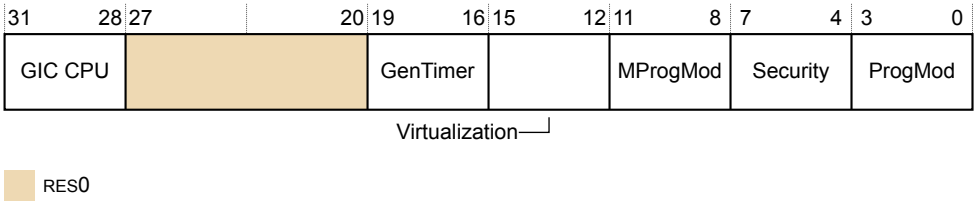


Figure B1-56 ID_PFR1 bit assignments

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 *B2.74 ID_PFR1_EL1, AArch32 Processor Feature Register 1, EL1* on page B2-392.

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 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.

- *B1.72.1 IFSR with Short-descriptor translation table format* on page B1-231.
- *B1.72.2 IFSR with Long-descriptor translation table format* on page B1-231.

Configurations

IFSR (NS) is architecturally mapped to AArch64 register IFSR32_EL2. See *B2.75 IFSR32_EL2, Instruction Fault Status Register, EL2* on page B2-394.

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:

- *B1.72.1 IFSR with Short-descriptor translation table format* on page B1-231.
- *B1.72.2 IFSR with Long-descriptor translation table format* on page B1-231.

B1.72.1 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:

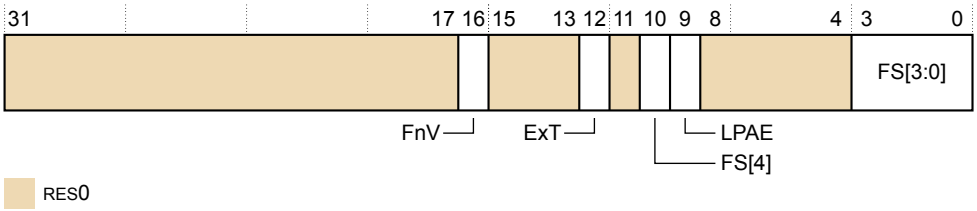


Figure B1-57 IFSR bit assignments for Short-descriptor translation table format

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*.

B1.72.2 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:

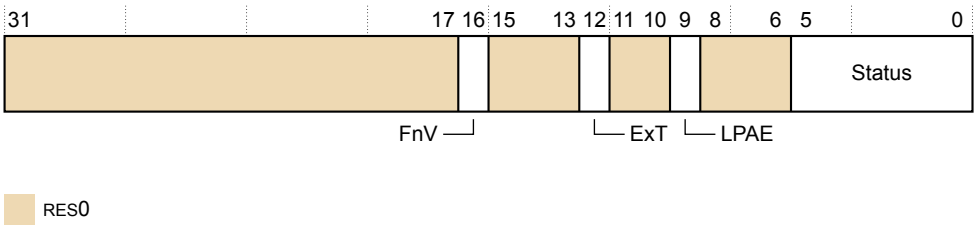


Figure B1-58 IFSR bit assignments for Long-descriptor translation table format

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*.

B1.73 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.

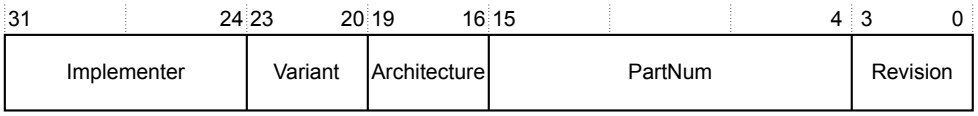


Figure B1-59 MIDR bit assignments

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:

`0x1` r1p0.

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:

`0xD05` Cortex-A55 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:

`0x0` r1p0.

Configurations
MIDR is:

- Architecturally mapped to the AArch64 MIDR_EL1 register. See *B2.82 MIDR_EL1, Main ID Register, EL1* on page B2-403.
- 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*.

B1.74 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.

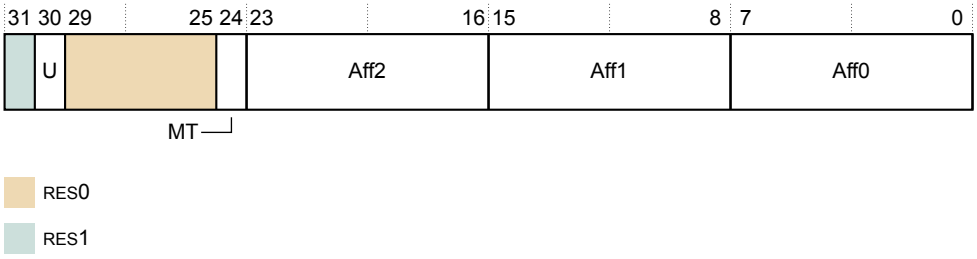


Figure B1-60 MPIDR bit assignments

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-A55 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-A55 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 *B2.83 MPIDR_EL1, Multiprocessor Affinity Register, EL1* on page B2-404.
- 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*.

B1.75 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 *B2.84 PAR_EL1, Physical Address Register, EL1* on page B2-406.

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:

- *B1.75.1 PAR with Short-descriptor translation table format* on page B1-236.
- *B1.75.2 PAR with Long-descriptor translation table format* on page B1-238.

B1.75.1 PAR with Short-descriptor translation table format

PAR details when the PE 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.

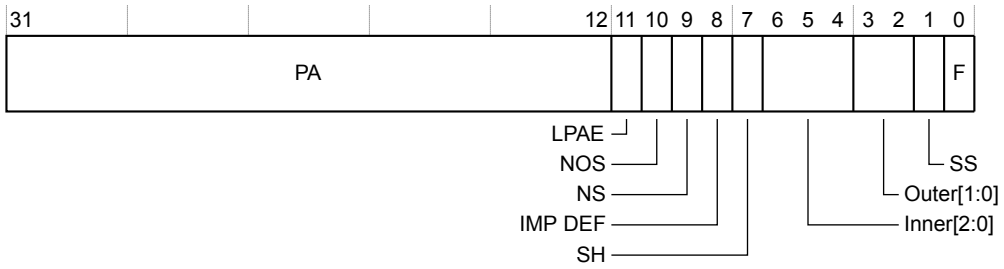


Figure B1-61 PAR bit assignments, PAR.F is 0

PA, [31:12]

Output address. The output address (OA) corresponding to the supplied input address. This field returns address bits[31:12].

LPAE, [11]

`0` Short-descriptor translation table format used. This means that the PAR returned a 32-bit value.

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.

`1` Memory region is Inner Shareable.

NS, [9]

Non-secure. The NS attribute for a translation table entry from a Secure translation regime.

For a result from a Secure translation regime, this bit reflects the Security state of the physical address space of the translation. This means it reflects the effect of the NSTable bits of earlier levels of the translation table walk if those NSTable bits effect the translation.

IMP DEF, [8]

IMPLEMENTATION DEFINED. Bit[8] is RES0. See the *ARM® Architecture Reference Manual ARMv8, for ARMv8-A architecture profile* for more information.

SH, [7]

Shareability. Indicates whether the physical memory region is Non-shareable:

`0` Memory is Non-shareable.

`1` Memory is shareable, and PAR.NOS indicates whether the region is Outer Shareable or Inner Shareable.

Inner[2:0], [6:4]

Inner cacheability attribute for the region. Permitted values are:

`000` Non-cacheable.

`001` Device-nGnRnE.

`011` Device-nGnRE.

`101` Write-Back, Write-Allocate.

`110` Write-Through.

`111` Write-Back, no Write-Allocate.

The values 010 and 100 are reserved.

Outer[1:0], [3:2]

`00` Non-cacheable.

`01` Write-Back, Write-Allocate.

`10` Write-Through, no Write-Allocate.

`11` Write-Back, no Write-Allocate.

SS, [1]

Supersection. Used to indicate if the result is a Supersection:

`0` Result is not a Supersection. PAR[31:12] contains OA[31:12].

`1` Result is a Supersection, and:

- PAR[31:24] contains OA[31:24].
- PAR[23:16] contains OA[39:32].
- PAR[15:12] contains `0b0000`.

If an implementation supports less than 40 bits of physical address, the bits in the PAR field that correspond to physical address bits that are not implemented are UNKNOWN

F, [0]

Indicates whether the instruction performed a successful address translation.

`0` Address translation completed successfully.

Bit field descriptions, PAR.F is 1

The following figure shows the PAR bit assignments when PAR.F is 1.

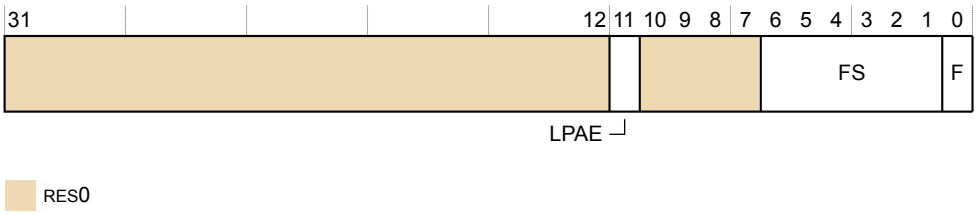


Figure B1-62 PAR bit assignments, PAR.F is 1

RES0, [31:12]

RES0 Reserved.

LPAE, [11]

0 Short-descriptor translation table format used. This means that the PAR returned a 32-bit value.

RES0, [10:7]

RES0 Reserved.

FS, [6:1]

Fault status bits. Bits [12,10,3:0] from the DFSR, indicating the source of the abort.

F, [0]

Indicates whether the instruction performed a successful address translation.

1 Address translation aborted.

B1.75.2 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.

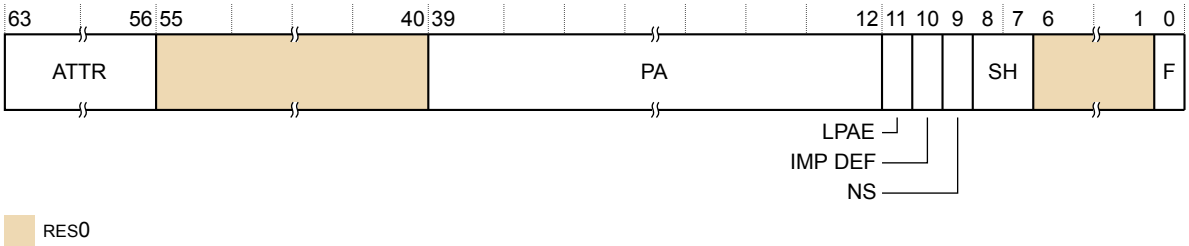


Figure B1-63 PAR bit assignments, PAR.F is 0

ATR, [63:56]

Memory attributes for the returned output address. This field uses the same encoding as the Attr<n> fields in MAIR0 and MAIR1.

RES0, [55:40]

RES0 Reserved.

PA, [39:12]
Output address. The output address (OA) corresponding to the supplied input address. This field returns address bits[31:12].

LPAE, [11]

`1` Long-descriptor translation table format used. This means that the PAR returned a 64-bit value.

IMP DEF, [10]
IMPLEMENTATION DEFINED.

NS, [9]
Non-secure. The NS attribute for a translation table entry from a Secure translation regime.

For a result from a Secure translation regime, this bit reflects the Security state of the physical address space of the translation. This means it reflects the effect of the NSTable bits of earlier levels of the translation table walk if those NSTable bits effect the translation.

For a result from a Non-secure translation regime, this bit is UNKNOWN.

SH, [8:7]
Shareability attribute, for the returned output address. Permitted values are:

`00` Non-shareable.

`10` Outer Shareable.

`11` Inner Shareable.The value 01 is reserved.

RES0, [6:1]

RES0 Reserved.

F, [0]
Indicates whether the instruction performed a successful address translation.

`0` Address translation completed successfully.

Bit field descriptions, PAR.F is 1

The following figure shows the PAR bit assignments when PAR.F is 1.

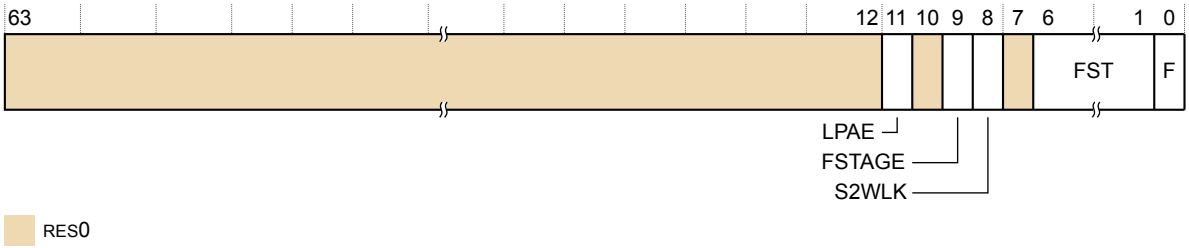


Figure B1-64 PAR bit assignments, PAR.F is 1

RES0, [63:12]

RES0 Reserved.

LPAE, [11]

`1` Long-descriptor translation table format used. This means the PAR returned a 64-bit value.

RES0, [10]

RES0 Reserved.

FSTAGE, [9]

Indicates the translation stage at which the translation aborted:

`0` Translation aborted because of a fault in the stage 1 translation.

`1` Translation aborted because of a fault in the stage 2 translation.

S2WLK, [8]

If this bit is set to 1, it indicates the translation aborted because of a stage 2 fault during a stage 1 translation table walk.

RES0, [7]

RES0 Reserved.

FST, [6:1]

Fault status field. Values are as in the DFSR.STATUS and IFSR.STATUS fields when using the Long-descriptor translation table format.

F, [0]

Indicates whether the instruction performed a successful address translation.

`1` Address translation aborted.

B1.76 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-A55 core.

Bit field descriptions

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

This register is Read Only.

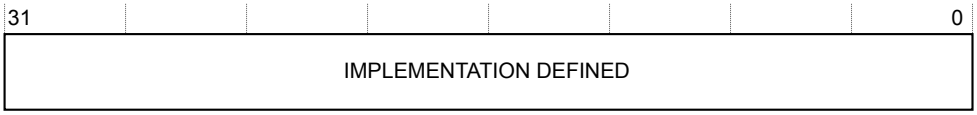


Figure B1-65 REVIDR bit assignments

IMPLEMENTATION DEFINED, [31:0]

Implementation Defined.

Configurations

AArch32 System register REVIDR is architecturally mapped to AArch64 System register REVIDR_EL1. See *B2.85 REVIDR_EL1, Revision ID Register, EL1* on page B2-407.

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

B1.77 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.

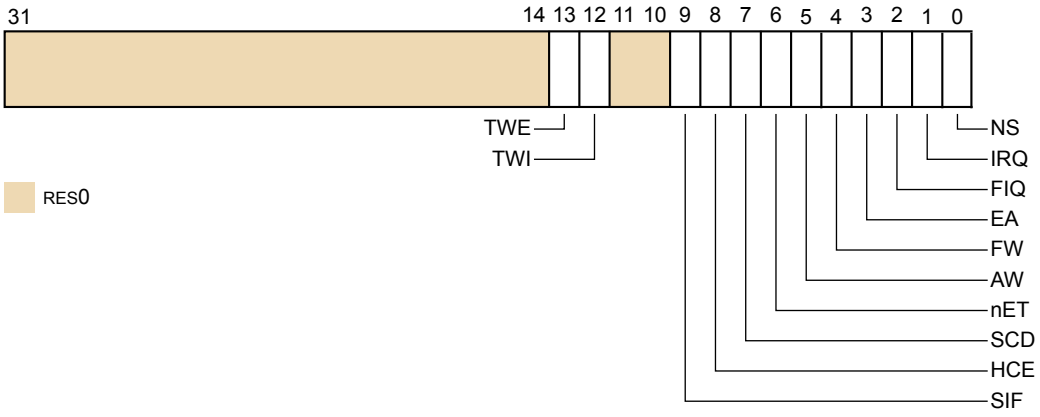


Figure B1-66 SCR bit assignments

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*.

B1.78 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.

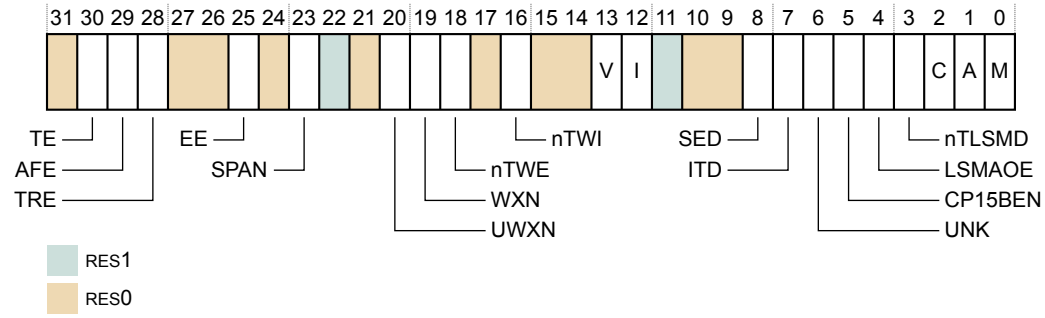


Figure B1-67 SCTLR bit assignments

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 *B2.87 SCTLR_EL1, System Control Register, EL1* on page B2-409.

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*.

B1.79 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.

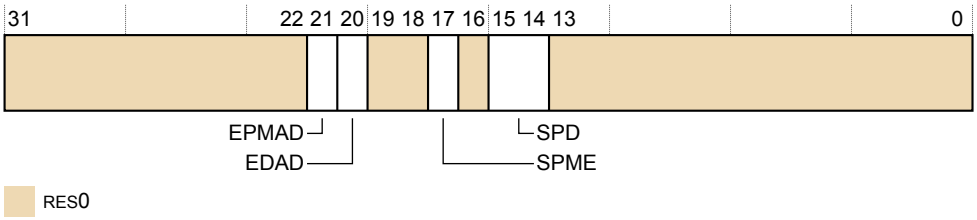


Figure B1-68 SDCR bit assignments

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 *B2.81 MDCR_EL3, Monitor Debug Configuration Register, EL3* on page B2-401.

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*.

B1.80 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.

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, and 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.

- *B1.80.1 TTBCR with Short-descriptor translation table format* on page B1-248.
- *B1.80.2 TTBCR with Long-descriptor translation table format* on page B1-249.

Configurations

TTBCR (NS) is architecturally mapped to AArch64 register TCR_EL1[31:0]. See *B2.90 TCR_EL1, Translation Control Register, EL1* on page B2-412.

TTBCR (S) is architecturally mapped to AArch64 register TCR_EL3[31:0]. See *B2.92 TCR_EL3, Translation Control Register, EL3* on page B2-417.

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:

- *B1.80.1 TTBCR with Short-descriptor translation table format* on page B1-248.
- *B1.80.2 TTBCR with Long-descriptor translation table format* on page B1-249.

B1.80.1 TTBCR with Short-descriptor translation table format

TTBCR has a specific format when using the Short-descriptor translation table format. TTBCR.EAE determines which format of the register is in use.

The following figure shows the TTBCR bit assignments when TTBCR.EAE is 0.

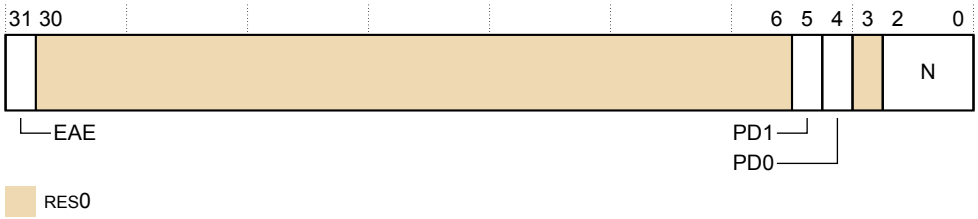


Figure B1-69 TTBCR bit assignments, TTBCR.EAE is 0

EAE, [31]

Extended Address Enable.

`0b0` Use the 32-bit translation system, with the Short-descriptor translation table format.

RES0, [30:6]

RES0 Reserved.

PD1, [5]

Translation table walk disable for translations using TTBR1. This bit controls whether a translation table walk is performed on a TLB miss, for an address that is translated using TTBR1. The possible values are:

`0b0` Perform translation table walks using TTBR1.

`0b1` A TLB miss on an address that is translated using TTBR1 generates a Translation fault. No translation table walk is performed.

PD0, [4]

Translation table walk disable for translations using TTBR0. This bit controls whether a translation table walk is performed on a TLB miss for an address that is translated using TTBR0. The possible values are:

`0b0` Perform translation table walks using TTBR0.

`0b1` A TLB miss on an address that is translated using TTBR0 generates a Translation fault. No translation table walk is performed.

RES0, [3]

RES0 Reserved.

N, [2:0]

Indicate the width of the base address held in TTBR0. In TTBR0, the base address field is bits[31:14-N]. The value of N also determines:

- Whether TTBR0 or TTBR1 is used as the base address for translation table walks.
- The size of the translation table pointed to by TTBR0.

N can take any value from 0 to 7, that is, from `0b000` to `0b111`.

When N has its reset value of 0, the translation table base is compatible with ARMv5 and ARMv6.

Resets to 0.

B1.80.2 TTBCR with Long-descriptor translation table format

TTBCR has a specific format when using the Long-descriptor translation table format. TTBCR.EAE determines which format of the register is in use.

The following figure shows the TTBCR bit assignments when TTBCR.EAE is 1.

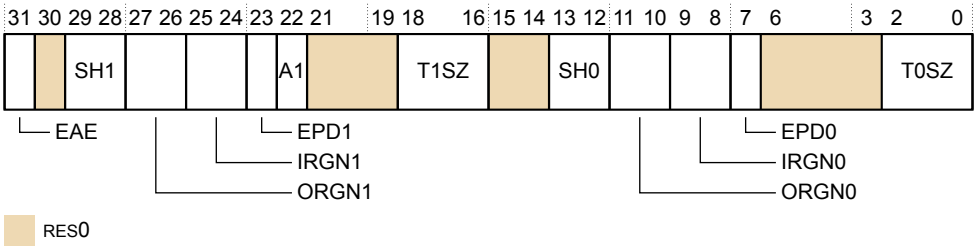


Figure B1-70 TTBCR bit assignments, TTBCR.EAE is 1

EAE, [31]

Extended Address Enable:

`0b1` Use the VMSAv8-32 translation system, with the Long-descriptor translation table format.

RES0, [30]

RES0 Reserved.

SH1, [29:28]

Shareability attribute for memory associated with translation table walks using TTBR1:

0b00 Non-shareable.

0b10 Outer Shareable.

0b11 Inner Shareable.

Other values are reserved.

Resets to 0.

ORGN1, [27:26]

Outer cacheability attribute for memory associated with translation table walks using TTBR1:

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.

Resets to 0.

IRGN1, [25:24]

Inner cacheability attribute for memory associated with translation table walks using TTBR1:

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.

Resets to 0.

EPD1, [23]

Translation table walk disable for translations using TTBR1. This bit controls whether a translation table walk is performed on a TLB miss, for an address that is translated using TTBR1:

0b0 Perform translation table walks using TTBR1.

0b1 A TLB miss on an address that is translated using TTBR1 generates a Translation fault. No translation table walk is performed.

A1, [22]

Selects whether TTBR0 or TTBR1 defines the ASID:

0b0 TTBR0.ASID defines the ASID.

0b1 TTBR1.ASID defines the ASID.

RES0, [21:19]

RES0 Reserved.

T1SZ, [18:16]

The size offset of the memory region addressed by TTBR1. The region size is $2^{32-T1SZ}$ bytes.

Resets to 0.

RES0, [15:14]

RES0 Reserved.

SH0, [13:12]

Shareability attribute for memory associated with translation table walks using TTBR0:

0b00 Non-shareable.

0b10 Outer Shareable.

`0b11` Inner Shareable.

Other values are reserved.

Resets to 0.

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.

Resets to 0.

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.

Resets to 0.

EPD0, [7]

Translation table walk disable for translations using TTBR0. This bit controls whether a translation table walk is performed on a TLB miss, for an address that is translated using TTBR0:

`0b0` Perform translation table walks using TTBR0.

`0b1` A TLB miss on an address that is translated using TTBR0 generates a Translation fault. No translation table walk is performed.

RES0, [6:3]

RES0 Reserved.

T0SZ, [2:0]

The size offset of the memory region addressed by TTBR0. The region size is $2^{32-T0SZ}$ bytes.

Resets to 0.

B1.81 TTBCR2, Translation Table Base Control Register 2

The TTBCR2 indicates the hierarchical permission disable and the page based hardware attributes.

Bit field descriptions

TTBCR2 is a 32-bit register, and is part of the Virtual memory control registers functional group.

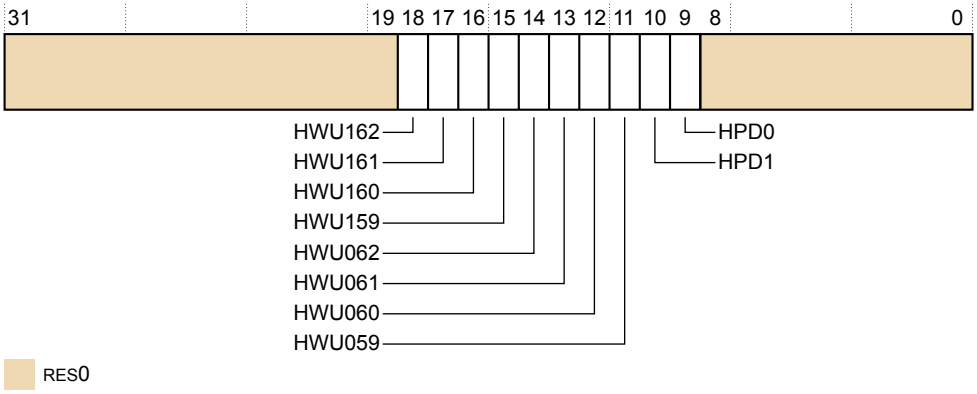


Figure B1-71 TTBCR2 bit assignments

RES0, [31:19]

RES0 Reserved.

HWU162, [18]

Indicates implementation defined hardware use of bit[62] of the stage1 translation table block or level 3 entry. The possible values are:

0 The associated stage 1 translation table entry bit cannot be interpreted by hardware for an implementation defined purpose.

1 The associated stage 1 translation table entry bit can be interpreted by hardware for an implementation defined purpose if the associated TTBCR2.HPD1 ==1.

HWU161, [17]

Indicates implementation defined hardware use of bit[61] of the stage1 translation table block or level 3 entry. The possible values are:

0 The associated stage 1 translation table entry bit cannot be interpreted by hardware for an implementation defined purpose.

1 The associated stage 1 translation table entry bit can be interpreted by hardware for an implementation defined purpose if the associated TTBCR2.HPD1 ==1.

HWU160, [16]

Indicates implementation defined hardware use of bit[60] of the stage1 translation table block or level 3 entry. The possible values are:

0 The associated stage 1 translation table entry bit cannot be interpreted by hardware for an implementation defined purpose.

1 The associated stage 1 translation table entry bit can be interpreted by hardware for an implementation defined purpose if the associated TTBCR2.HPD1 ==1.

HWU159, [15]

Indicates implementation defined hardware use of bit[59] of the stage1 translation table block or level 3 entry. The possible values are:

0 The associated stage 1 translation table entry bit cannot be interpreted by hardware for an implementation defined purpose.

1 The associated stage 1 translation table entry bit can be interpreted by hardware for an implementation defined purpose if the associated TTBCR2.HPD1 ==1.

HWU062, [14]

Indicates implementation defined hardware use of bit[62] of the stage1 translation table block or level 3 entry. The possible values are:

0 The associated stage 1 translation table entry bit cannot be interpreted by hardware for an implementation defined purpose.

1 The associated stage 1 translation table entry bit can be interpreted by hardware for an implementation defined purpose if the associated TTBCR2.HPD0 ==1.

HWU061, [13]

Indicates implementation defined hardware use of bit[61] of the stage1 translation table block or level 3 entry. The possible values are:

0 The associated stage 1 translation table entry bit cannot be interpreted by hardware for an implementation defined purpose.

1 The associated stage 1 translation table entry bit can be interpreted by hardware for an implementation defined purpose if the associated TTBCR2.HPD0 ==1.

HWU060, [12]

Indicates implementation defined hardware use of bit[60] of the stage1 translation table block or level 3 entry. The possible values are:

0 The associated stage 1 translation table entry bit cannot be interpreted by hardware for an implementation defined purpose.

1 The associated stage 1 translation table entry bit can be interpreted by hardware for an implementation defined purpose if the associated TTBCR2.HPD0 ==1.

HWU059, [11]

Indicates implementation defined hardware use of bit[59] of the stage1 translation table block or level 3 entry. The possible values are:

0 The associated stage 1 translation table entry bit cannot be interpreted by hardware for an implementation defined purpose.

1 The associated stage 1 translation table entry bit can be interpreted by hardware for an implementation defined purpose if the associated TTBCR2.HPD0 ==1.

HPD1, [10]

Hierarchical permission disable 1. The possible values are:

0 Hierarchical permissions for the TTBR1 region are enabled.

1 Hierarchical permissions for the TTBR1 region are disabled if TTBCR.T2E is set to 1. If TTBCR.T2E is set to 0, hierarchical permissions are enabled.

HPD0, [9]

Hierarchical permission disable 0. The possible values are:

0 Hierarchical permissions for the TTBR0 region are enabled.

1 Hierarchical permissions for the TTBR0 region are disabled if TTBCR.T2E is set to 1. If TTBCR.T2E is set to 0, hierarchical permissions are enabled.

RES0, [8:0]

RES0 Reserved.

Configurations TTBCR2 (NS) is architecturally mapped to AArch64 register TCR_EL1. See *B2.90 TCR_EL1, Translation Control Register, EL1* on page B2-412.

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*.

B1.82 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.

- *B1.82.1 TTBR0 with Short-descriptor translation table format* on page B1-255.
- *B1.82.2 TTBR0 with Long-descriptor translation table format* on page B1-256.

Configurations

TTBR0 (NS) is architecturally mapped to AArch64 register TTBR0_EL1. See *B2.93 TTBR0_EL1, Translation Table Base Register 0, EL1* on page B2-419.

TTBR0 (S) is mapped to AArch64 register TTBR0_EL3. See *B2.95 TTBR0_EL3, Translation Table Base Register 0, EL3* on page B2-421.

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:

- *B1.82.1 TTBR0 with Short-descriptor translation table format* on page B1-255.
- *B1.82.2 TTBR0 with Long-descriptor translation table format* on page B1-256.

B1.82.1 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.

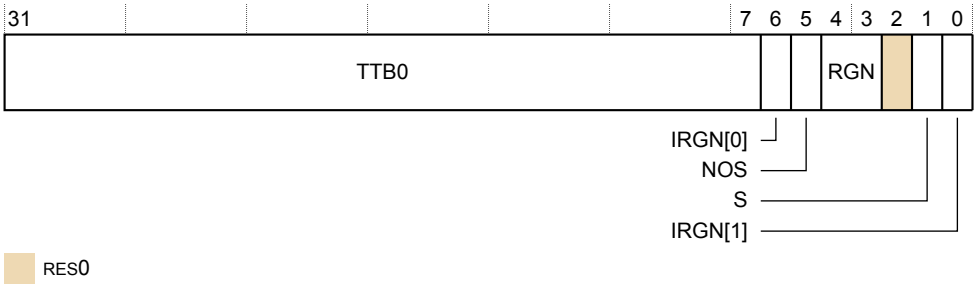


Figure B1-72 TTBR0 bit assignments, TTBCR.EAE is 0

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*.

B1.82.2 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.

B1.83 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.

- *B1.83.1 TTBR1 with Short-descriptor translation table format* on page B1-257.
- *B1.83.2 TTBR1 with Long-descriptor translation table format* on page B1-258.

Configurations

TTBR1 (NS) is architecturally mapped to AArch64 register TTBR1_EL1. See *B2.96 TTBR1_EL1, Translation Table Base Register 1, EL1* on page B2-422.

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:

- *B1.83.1 TTBR1 with Short-descriptor translation table format* on page B1-257.
- *B1.83.2 TTBR1 with Long-descriptor translation table format* on page B1-258.

B1.83.1 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.

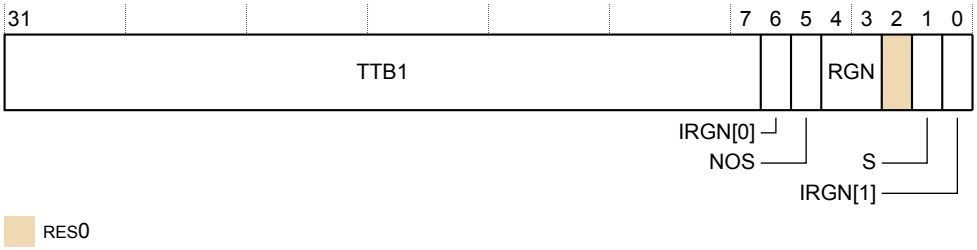


Figure B1-73 TTBR1 bit assignments, TTBCR.EAE is 0

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*.

B1.83.2 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.

B1.84 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.

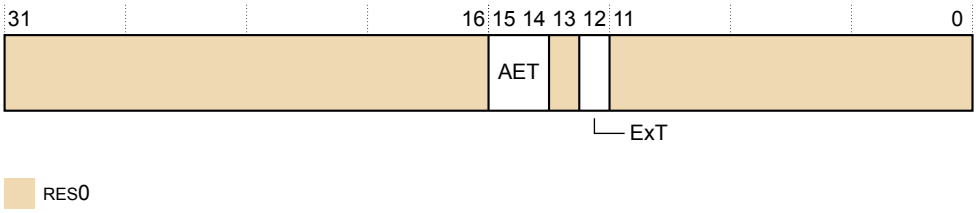


Figure B1-74 VDFSR bit assignments

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:

`0b01` *Uncorrected error, Unrecoverable error* (UEU).

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 B2-429.

Bit fields and details not provided in this description are architecturally defined. See the *ARM® Architecture Reference Manual ARMv8, for ARMv8-A architecture profile*.

B1.85 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 *B1.85.1 VDISR with Short-descriptor translation table format* on page B1-260
- When written at EL1 using long-descriptor format. See *B1.85.2 VDISR with Long-descriptor translation table format* on page B1-261

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 *B2.98 VDISR_EL2, Virtual Deferred Interrupt Status Register, EL2* on page B2-424.

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 VDISR with Short-descriptor translation table format* on page B1-260.
- *B1.85.2 VDISR with Long-descriptor translation table format* on page B1-261.

B1.85.1 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.

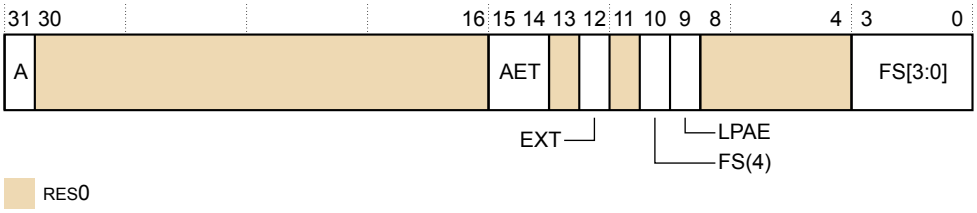


Figure B1-75 VDISR bit assignments for Short-descriptor translation table format

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:

`0b01` *Uncorrected error, Unrecoverable error* (UEU).

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.

B1.85.2 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.

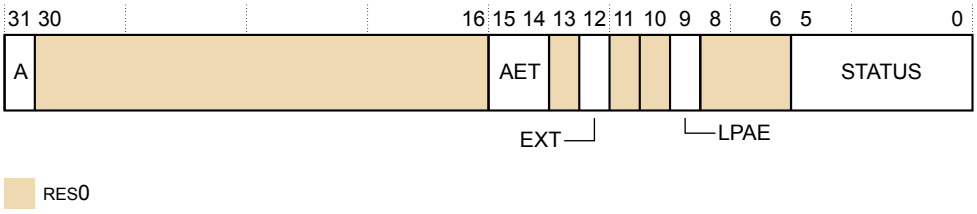


Figure B1-76 VDISR bit assignments for Long-descriptor translation table format

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:

`0b01` *Uncorrected error, Unrecoverable error* (UEU).

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.

B1.86 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*.

B1.87 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*.

B1.88 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.

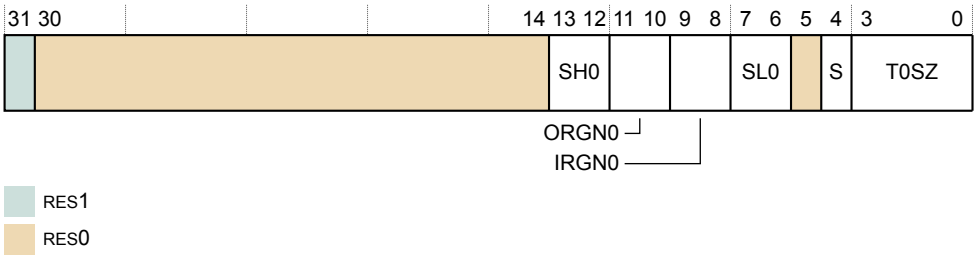


Figure B1-77 VTCR bit assignments

RES1, [31]

RES1 Reserved.

RES0, [30:14]

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 B2-431.

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 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.

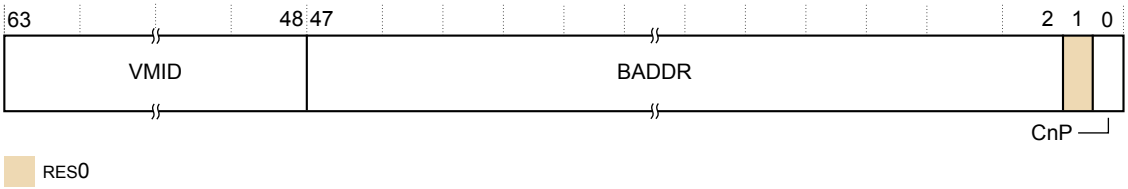


Figure B1-78 VTTBR bit assignments

CnP, [0]

Common not Private. The reset value is:

0 CnP is not supported.

Configurations

VTTBR is architecturally mapped to AArch64 register VTTBR_EL1. See *B2.103 VTTBR_EL2, Virtualization Translation Table Base Register, EL2* on page B2-432.

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-A55 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-A55 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-A55 core.

The section contains two tables:

Registers with implementation defined bit fields

This table identifies the architecturally defined registers in Cortex-A55 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 B2-273.

Other architecturally defined registers

This table identifies the other architecturally defined registers that are implemented in the Cortex-A55 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 B2-277.

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 B2-289
ACTLR_EL2	3	c1	4	c0	1	64	*B2.6 ACTLR_EL2, Auxiliary Control Register, EL2* on page B2-290
ACTLR_EL3	3	c1	6	c0	1	64	*B2.7 ACTLR_EL3, Auxiliary Control Register, EL3* on page B2-292
AIDR_EL1	3	c0	1	c0	7	32	*B2.14 AIDR_EL1, Auxiliary ID Register, EL1* on page B2-300
AFSR0_EL1	3	c5	0	c1	0	32	*B2.8 AFSR0_EL1, Auxiliary Fault Status Register 0, EL1* on page B2-294
AFSR0_EL2	3	c5	4	c1	0	32	*B2.9 AFSR0_EL2, Auxiliary Fault Status Register 0, EL2* on page B2-295
AFSR0_EL3	3	c5	6	c1	0	32	*B2.10 AFSR0_EL3, Auxiliary Fault Status Register 0, EL3* on page B2-296
AFSR1_EL1	3	c5	0	c1	1	32	*B2.11 AFSR1_EL1, Auxiliary Fault Status Register 1, EL1* on page B2-297
AFSR1_EL2	3	c5	4	c1	1	32	*B2.12 AFSR1_EL2, Auxiliary Fault Status Register 1, EL2* on page B2-298
AFSR1_EL3	3	c5	6	c1	1	32	*B2.13 AFSR1_EL3, Auxiliary Fault Status Register 1, EL3* on page B2-299
AMAIR_EL1	3	c10	0	c3	0	64	*B2.15 AMAIR_EL1, Auxiliary Memory Attribute Indirection Register, EL1* on page B2-301
AMAIR_EL2	3	c10	4	c3	0	64	*B2.16 AMAIR_EL2, Auxiliary Memory Attribute Indirection Register, EL2* on page B2-302
AMAIR_EL3	3	c10	6	c3	0	64	*B2.17 AMAIR_EL3, Auxiliary Memory Attribute Indirection Register, EL3* on page B2-303
CCSIDR_EL1	3	c0	1	c0	0	32	*B2.18 CCSIDR_EL1, Cache Size ID Register, EL1* on page B2-304

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 B2-306
CPACR_EL1	3	c1	0	c0	2	32	*B2.20 CPACR_EL1, Architectural Feature Access Control Register, EL1* on page B2-308
CPTR_EL2	3	c1	4	c1	2	32	*B2.21 CPTR_EL2, Architectural Feature Trap Register, EL2* on page B2-309
CPTR_EL3	3	c1	6	c1	2	32	*B2.22 CPTR_EL3, Architectural Feature Trap Register, EL3* on page B2-310
CSSELR_EL1	3	c0	2	c0	0	32	*B2.31 CSSELR_EL1, Cache Size Selection Register, EL1* on page B2-328
CTR_EL0	3	c0	3	c0	1	32	*B2.32 CTR_EL0, Cache Type Register, EL0* on page B2-329
DISR_EL1	3	c12	0	c1	1	64	*B2.34 DISR_EL1, Deferred Interrupt Status Register, EL1* on page B2-332
ERRIDR_EL1	3	c5	0	c3	0	32	*B2.35 ERRIDR_EL1, Error ID Register, EL1* on page B2-334
ERRSELR_EL1	3	c5	0	c3	1	32	*B2.36 ERRSELR_EL1, Error Record Select Register, EL1* on page B2-335
ERXADDR_EL1	3	c5	0	c4	3	64	*B2.37 ERXADDR_EL1, Selected Error Record Address Register, EL1* on page B2-336
ERXCTLR_EL1	3	c5	0	c4	1	64	*B2.38 ERXCTLR_EL1, Selected Error Record Control Register, EL1* on page B2-337
ERXFR_EL1	3	c5	0	c4	0	64	*B2.39 ERXFR_EL1, Selected Error Record Feature Register, EL1* on page B2-338
ERXMISC0_EL1	3	c5	0	c5	0	64	*B2.40 ERXMISC0_EL1, Selected Error Record Miscellaneous Register 0, EL1* on page B2-339
ERXMISC1_EL1	3	c5	0	c5	1	64	*B2.41 ERXMISC1_EL1, Selected Error Record Miscellaneous Register 1, EL1* on page B2-340
ERXSTATUS_EL1	3	c5	0	c4	2	32	*B2.45 ERXSTATUS_EL1, Selected Error Record Primary Status Register, EL1* on page B2-346
ESR_EL1	3	c5	0	c2	0	32	*B2.46 ESR_EL1, Exception Syndrome Register, EL1* on page B2-347
ESR_EL2	3	c5	4	c2	0	32	*B2.47 ESR_EL2, Exception Syndrome Register, EL2* on page B2-348
ESR_EL3	3	c5	6	c2	0	32	*B2.48 ESR_EL3, Exception Syndrome Register, EL3* on page B2-349
HACR_EL2	3	c1	4	c1	7	32	*B2.49 HACR_EL2, Hyp Auxiliary Configuration Register, EL2* on page B2-350
HCR_EL2	3	c1	4	c1	0	64	*B2.50 HCR_EL2, Hypervisor Configuration Register, EL2* on page B2-351
ID_AFR0_EL1	3	c0	0	c1	3	32	*B2.59 ID_AFR0_EL1, AArch32 Auxiliary Feature Register 0, EL1* on page B2-365

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	*B2.60 ID_DFR0_EL1, AArch32 Debug Feature Register 0, EL1* on page B2-366
ID_ISAR0_EL1	3	c0	0	c2	0	32	*B2.61 ID_ISAR0_EL1, AArch32 Instruction Set Attribute Register 0, EL1* on page B2-368
ID_ISAR1_EL1	3	c0	0	c2	1	32	*B2.62 ID_ISAR1_EL1, AArch32 Instruction Set Attribute Register 1, EL1* on page B2-370
ID_ISAR2_EL1	3	c0	0	c2	2	32	*B2.63 ID_ISAR2_EL1, AArch32 Instruction Set Attribute Register 2, EL1* on page B2-372
ID_ISAR3_EL1	3	c0	0	c2	3	32	*B2.64 ID_ISAR3_EL1, AArch32 Instruction Set Attribute Register 3, EL1* on page B2-374
ID_ISAR4_EL1	3	c0	0	c2	4	32	*B2.65 ID_ISAR4_EL1, AArch32 Instruction Set Attribute Register 4, EL1* on page B2-376
ID_ISAR5_EL1	3	c0	0	c2	5	32	*B2.66 ID_ISAR5_EL1, AArch32 Instruction Set Attribute Register 5, EL1* on page B2-378
ID_ISAR6_EL1	3	c0	0	c2	7	32	*B2.67 ID_ISAR6_EL1, AArch32 Instruction Set Attribute Register 6, EL1* on page B2-380
ID_MMFR0_EL1	3	c0	0	c1	4	32	*B2.68 ID_MMFR0_EL1, AArch32 Memory Model Feature Register 0, EL1* on page B2-381
ID_MMFR1_EL1	3	c0	0	c1	5	32	*B2.69 ID_MMFR1_EL1, AArch32 Memory Model Feature Register 1, EL1* on page B2-383
ID_MMFR2_EL1	3	c0	0	c1	6	32	*B2.70 ID_MMFR2_EL1, AArch32 Memory Model Feature Register 2, EL1* on page B2-385
ID_MMFR3_EL1	3	c0	0	c1	7	32	*B2.71 ID_MMFR3_EL1, AArch32 Memory Model Feature Register 3, EL1* on page B2-387
ID_MMFR4_EL1	3	c0	0	c2	6	32	*B2.72 ID_MMFR4_EL1, AArch32 Memory Model Feature Register 4, EL1* on page B2-389
ID_PFR0_EL1	3	c0	0	c1	0	32	*B2.73 ID_PFR0_EL1, AArch32 Processor Feature Register 0, EL1* on page B2-391
ID_PFR1_EL1	3	c0	0	c1	1	32	*B2.74 ID_PFR1_EL1, AArch32 Processor Feature Register 1, EL1* on page B2-392
ID_AA64DFR0_EL1	3	c0	0	c5	0	64	*B2.52 ID_AA64DFR0_EL1, AArch64 Debug Feature Register 0, EL1* on page B2-354
ID_AA64ISAR0_EL1	3	c0	0	c6	0	64	*B2.53 ID_AA64ISAR0_EL1, AArch64 Instruction Set Attribute Register 0, EL1* on page B2-356
ID_AA64ISAR1_EL1	3	c0	0	c6	1	64	*B2.54 ID_AA64ISAR1_EL1, AArch64 Instruction Set Attribute Register 1, EL1* on page B2-358
ID_AA64MMFR0_EL1	3	c0	0	c7	0	64	*B2.55 ID_AA64MMFR0_EL1, AArch64 Memory Model Feature Register 0, EL1* on page B2-359
ID_AA64MMFR1_EL1	3	c0	0	c7	1	64	*B2.56 ID_AA64MMFR1_EL1, AArch64 Memory Model Feature Register 1, EL1* on page B2-360
ID_AA64MMFR2_EL1	3	c0	0	c7	2	64	*B2.57 ID_AA64MMFR2_EL1, AArch64 Memory Model Feature Register 2, EL1* on page B2-362

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	*B2.58 ID_AA64PFR0_EL1, AArch64 Processor Feature Register 0, EL1* on page B2-363
IFSR32_EL2	3	c5	4	c0	1	32	*B2.75 IFSR32_EL2, Instruction Fault Status Register, EL2* on page B2-394
LORC_EL1	3	c10	0	c4	3	64	*B2.76 LORC_EL1, LORegion Control Register, EL1* on page B2-396
LORID_EL1	3	c10	0	c4	7	64	*B2.78 LORID_EL1, Limited Order Region Identification Register, EL1* on page B2-398
LORN_EL1	3	c10	0	c4	2	64	*B2.79 LORN_EL1, LORegion Number Register, EL1* on page B2-399
MDCR_EL3	3	c1	6	c3	1	32	*B2.81 MDCR_EL3, Monitor Debug Configuration Register, EL3* on page B2-401
MIDR_EL1	3	c0	0	c0	0	32	*B2.82 MIDR_EL1, Main ID Register, EL1* on page B2-403
MPIDR_EL1	3	c0	0	c0	5	64	*B2.83 MPIDR_EL1, Multiprocessor Affinity Register, EL1* on page B2-404
PAR_EL1	3	c7	0	c4	0	64	*B2.84 PAR_EL1, Physical Address Register, EL1* on page B2-406
RVBAR_EL3	3	c12	6	c0	1	64	*B2.86 RVBAR_EL3, Reset Vector Base Address Register, EL3* on page B2-408
REVIDR_EL1	3	c0	0	c0	6	32	*B2.85 REVIDR_EL1, Revision ID Register, EL1* on page B2-407
SCTLR_EL1	3	c1	0	c0	0	32	*B2.87 SCTLR_EL1, System Control Register, EL1* on page B2-409
SCTLR_EL3	3	c1	6	c0	0	32	*B2.89 SCTLR_EL3, System Control Register, EL3* on page B2-411
TCR_EL1	3	c2	0	c0	2	64	*B2.90 TCR_EL1, Translation Control Register, EL1* on page B2-412
TCR_EL2	3	c2	4	c0	2	64	*B2.91 TCR_EL2, Translation Control Register, EL2* on page B2-413
TCR_EL3	3	c2	6	c0	2	64	*B2.92 TCR_EL3, Translation Control Register, EL3* on page B2-417
TTBR0_EL1	3	c2	0	c0	0	64	*B2.93 TTBR0_EL1, Translation Table Base Register 0, EL1* on page B2-419
TTBR0_EL2	3	c2	4	c0	0	64	*B2.94 TTBR0_EL2, Translation Table Base Register 0, EL2* on page B2-420
TTBR0_EL3	3	c2	6	c0	0	64	*B2.95 TTBR0_EL3, Translation Table Base Register 0, EL3* on page B2-421
TTBR1_EL1	3	c2	0	c0	1	64	*B2.96 TTBR1_EL1, Translation Table Base Register 1, EL1* on page B2-422
TTBR1_EL2	3	c2	4	c0	1	64	*B2.97 TTBR1_EL2, Translation Table Base Register 1, EL2* on page B2-423
VDISR_EL2	3	c12	4	c1	1	64	*B2.98 VDISR_EL2, Virtual Deferred Interrupt Status Register, EL2* on page B2-424

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 B2-429
VTCR_EL2	3	c2	4	c1	2	32	*B2.102 VTCR_EL2, Virtualization Translation Control Register, EL2* on page B2-431
VTTBR_EL2	3	c2	4	c1	0	64	*B2.103 VTTBR_EL2, Virtualization Translation Table Base Register, EL2* on page B2-432

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 B2-311
CPUCFR_EL1	3	c15	0	c0	0	32	*B2.24 CPUCFR_EL1, CPU Configuration Register, EL1* on page B2-313
CPUECTLR_EL1	3	c15	0	c1	4	64	*B2.25 CPUECTLR_EL1, CPU Extended Control Register, EL1* on page B2-315
CPUPCR_EL3	3	15	6	c8	1	64	*B2.26 CPUPCR_EL3, CPU Private Control Register, EL3* on page B2-318
CPUPMR_EL3	3	c15	6	c8	3	64	*B2.27 CPUPMR_EL3, CPU Private Mask Register, EL3* on page B2-320
CPUPOR_EL3	3	c15	6	c8	2	64	*B2.28 CPUPOR_EL3, CPU Private Operation Register, EL3* on page B2-322
CPUPSELR_EL3	3	c15	6	c8	0	32	*B2.29 CPUPSELR_EL3, CPU Private Selection Register, EL3* on page B2-324
CPUPWRCTLR_EL1	3	c15	0	c2	7	32	*B2.30 CPUPWRCTLR_EL1, Power Control Register, EL1* on page B2-326
ERXPFGCDNR_EL1	3	c15	0	c2	2	32	*B2.42 ERXPFGCDNR_EL1, Selected Error Pseudo Fault Generation Count Down Register, EL1* on page B2-341
ERXPFGCTLR_EL1	3	c15	0	c2	1	32	*B2.43 ERXPFGCTLR_EL1, Selected Error Pseudo Fault Generation Control Register, EL1* on page B2-343
ERXPFGFR_EL1	3	c15	0	c2	0	32	*B2.44 ERXPFGFR_EL1, Selected Pseudo Fault Generation Feature Register, EL1* on page B2-345

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.
CLUSTERTHREADSID_EL1	3	c15	0	c4	0	32-bit	Cluster thread scheme ID register.

Table B2-4 Cluster registers (continued)

Name	Op0	CRn	op1	CRm	op2	Width	Description
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 or 64-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 B2-300
CCSIDR__EL1	RO	-	*B2.18 CCSIDR_EL1, Cache Size ID Register, EL1* on page B2-304
CLIDR_EL1	RO	UNK Unknown: [31:30], [25:24], [22:21], 8, 5 'b1: [1:0] 'b0: [29:26], 23, [20:9], [7:6], [4:2]	*B2.19 CLIDR_EL1, Cache Level ID Register, EL1* on page B2-306 If the L2 cache is not implemented, the value is `0x09200003`.
CSSELR_EL1	RW	`0x00000000`	*B2.31 CSSELR_EL1, Cache Size Selection Register, EL1* on page B2-328
CTR_EL0	RO	`0x84448004`	*B2.32 CTR_EL0, Cache Type Register, EL0* on page B2-329
DCZID_EL0	RO	-	*B2.33 DCZID_EL0, Data Cache Zero ID Register, EL0* on page B2-331
ERRIDR_EL1	RO	-	*B2.35 ERRIDR_EL1, Error ID Register, EL1* on page B2-334
ID_AA64DFR0_EL1	RO	`0x0000000010305408`	*B2.52 ID_AA64DFR0_EL1, AArch64 Debug Feature Register 0, EL1* on page B2-354
ID_AA64ISAR0_EL1	RO	UNK Unknown: 12, 8, 5 'b1: 28, 21, 16 'b0: [64:29], [27:22], [20:17], [15:13], [11:9], [7:6], [4:0]	*B2.53 ID_AA64ISAR0_EL1, AArch64 Instruction Set Attribute Register 0, EL1* on page B2-356 If the Cryptographic extensions are not implemented or disabled these fields will read `0`.
ID_AA64ISAR1_EL1	RO	`0x0000000000100001`	*B2.54 ID_AA64ISAR1_EL1, AArch64 Instruction Set Attribute Register 1, EL1* on page B2-358
ID_AA64MMFR0_EL1	RO	`0x0000000000101122`	*B2.55 ID_AA64MMFR0_EL1, AArch64 Memory Model Feature Register 0, EL1* on page B2-359
ID_AA64MMFR1_EL1	RO	`0x0000000010212122`	*B2.56 ID_AA64MMFR1_EL1, AArch64 Memory Model Feature Register 1, EL1* on page B2-360
ID_AA64MMFR2_EL1	RO	`0x0000000000001011`	*B2.57 ID_AA64MMFR2_EL1, AArch64 Memory Model Feature Register 2, EL1* on page B2-362
ID_AA64PFR0_EL1	RO	UNK Unknown: 24, [23:20], [19:16] 'b1: 28, 13, 9, 5, 1 'b0: [63:29], [27:25], [15:14], [12:10], [8:6], [4:2], 0	*B2.58 ID_AA64PFR0_EL1, AArch64 Processor Feature Register 0, EL1* on page B2-363 If Advanced-SIMD/FP has been configured then [23:20] and [19:16] report `0001`, otherwise they report `1111`. If the GICv4 interface is disabled (GICCDISABLE is high) [24] will read `0`, otherwise it will read `1`.

(continued)

Name	Type	Reset	Description
ID_AFR0_EL1	RO	0x00000000	*B2.59 ID_AFR0_EL1, AArch32 Auxiliary Feature Register 0, EL1* on page B2-365
ID_DFR0_EL1	RO	0x04010088	*B2.60 ID_DFR0_EL1, AArch32 Debug Feature Register 0, EL1* on page B2-366 Bits [19:16] are 0x1 if ETM is implemented, and 0x0 otherwise.
ID_ISAR0_EL1	RO	0x02101110	*B2.61 ID_ISAR0_EL1, AArch32 Instruction Set Attribute Register 0, EL1* on page B2-368
ID_ISAR1_EL1	RO	0x13112111	*B2.62 ID_ISAR1_EL1, AArch32 Instruction Set Attribute Register 1, EL1* on page B2-370
ID_ISAR2_EL1	RO	0x21232042	*B2.63 ID_ISAR2_EL1, AArch32 Instruction Set Attribute Register 2, EL1* on page B2-372
ID_ISAR3_EL1	RO	0x01112131	*B2.64 ID_ISAR3_EL1, AArch32 Instruction Set Attribute Register 3, EL1* on page B2-374
ID_ISAR4_EL1	RO	0x00011142	*B2.65 ID_ISAR4_EL1, AArch32 Instruction Set Attribute Register 4, EL1* on page B2-376
ID_ISAR5_EL1	RO	0x00011121	*B2.66 ID_ISAR5_EL1, AArch32 Instruction Set Attribute Register 5, EL1* on page B2-378 ID_ISAR5 has the value 0x00010001 if the Cryptographic Extension is not implemented and enabled.
ID_ISAR6_EL1	RO	0x00000010	*B2.67 ID_ISAR6_EL1, AArch32 Instruction Set Attribute Register 6, EL1* on page B2-380
ID_MMFR0_EL1	RO	0x10201105	*B2.68 ID_MMFR0_EL1, AArch32 Memory Model Feature Register 0, EL1* on page B2-381
ID_MMFR1_EL1	RO	0x40000000	*B2.69 ID_MMFR1_EL1, AArch32 Memory Model Feature Register 1, EL1* on page B2-383
ID_MMFR2_EL1	RO	0x01260000	*B2.70 ID_MMFR2_EL1, AArch32 Memory Model Feature Register 2, EL1* on page B2-385
ID_MMFR3_EL1	RO	0x02122211	*B2.71 ID_MMFR3_EL1, AArch32 Memory Model Feature Register 3, EL1* on page B2-387
ID_MMFR4_EL1	RO	0x00021110	*B2.72 ID_MMFR4_EL1, AArch32 Memory Model Feature Register 4, EL1* on page B2-389
ID_PFR0_EL1	RO	0x00000131	*B2.73 ID_PFR0_EL1, AArch32 Processor Feature Register 0, EL1* on page B2-391
ID_PFR1_EL1	RO	0x10011011	*B2.74 ID_PFR1_EL1, AArch32 Processor Feature Register 1, EL1* on page B2-392 Bits [31:28] are 0x1 if the GIC CPU interface is implemented and enabled, and 0x0 otherwise.
LORID_EL1	RO	0x0000000000040004	*B2.78 LORID_EL1, Limited Order Region Identification Register, EL1* on page B2-398
MIDR_EL1	RO	0x411FD050	*B2.82 MIDR_EL1, Main ID Register, EL1* on page B2-403

(continued)

Name	Type	Reset	Description
MPIDR_EL1	RO	Unknown: [29:25], [23:16], [10:8] 'b1: 31, 24 'b0: [15:11], [7:0]	*B2.83 MPIDR_EL1, Multiprocessor Affinity Register, EL1* on page B2-404
REVIDR_EL1	RO	`0x00000000`	*B2.85 REVIDR_EL1, Revision ID Register, EL1* on page B2-407
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 B2-289
ACTLR_EL2	RW	*B2.6 ACTLR_EL2, Auxiliary Control Register, EL2* on page B2-290
ACTLR_EL3	RW	*B2.7 ACTLR_EL3, Auxiliary Control Register, EL3* on page B2-292
CPACR_EL1	RW	*B2.20 CPACR_EL1, Architectural Feature Access Control Register, EL1* on page B2-308
SCTLR_EL1	RW	*B2.87 SCTLR_EL1, System Control Register, EL1* on page B2-409
SCTLR_EL3	RW	*B2.89 SCTLR_EL3, System Control Register, EL3* on page B2-411

Reliability, Availability, Serviceability (RAS) registers

Name	Type	Description
DISR_EL1	RW	*B2.34 DISR_EL1, Deferred Interrupt Status Register, EL1* on page B2-332
ERRIDR_EL1	RW	*B2.35 ERRIDR_EL1, Error ID Register, EL1* on page B2-334
ERRSELR_EL1	RW	*B2.36 ERRSELR_EL1, Error Record Select Register, EL1* on page B2-335
ERXADDR_EL1	RW	*B2.37 ERXADDR_EL1, Selected Error Record Address Register, EL1* on page B2-336
ERXCTLR_EL1	RW	*B2.38 ERXCTLR_EL1, Selected Error Record Control Register, EL1* on page B2-337
ERXFR_EL1	RO	*B2.39 ERXFR_EL1, Selected Error Record Feature Register, EL1* on page B2-338
ERXMISC0_EL1	RW	*B2.40 ERXMISC0_EL1, Selected Error Record Miscellaneous Register 0, EL1* on page B2-339
ERXMISC1_EL1	RW	*B2.41 ERXMISC1_EL1, Selected Error Record Miscellaneous Register 1, EL1* on page B2-340
ERXSTATUS_EL1	RW	*B2.45 ERXSTATUS_EL1, Selected Error Record Primary Status Register, EL1* on page B2-346
ERXPFGCDNR_EL1	RW	*B2.42 ERXPFGCDNR_EL1, Selected Error Pseudo Fault Generation Count Down Register, EL1* on page B2-341
ERXPFCTLR_EL1	RW	*B2.43 ERXPFGCTLR_EL1, Selected Error Pseudo Fault Generation Control Register, EL1* on page B2-343
ERXPFGFR_EL1	RO	*B2.44 ERXPFGFR_EL1, Selected Pseudo Fault Generation Feature Register, EL1* on page B2-345
HCR_EL2	RW	*B2.50 HCR_EL2, Hypervisor Configuration Register, EL2* on page B2-351

(continued)

Name	Type	Description
VDISR_EL2	RW	*B2.98 VDISR_EL2, Virtual Deferred Interrupt Status Register, EL2* on page B2-424
VSESR_EL2	RW	*B2.101 VSESR_EL2, Virtual SError Exception Syndrome Register* on page B2-429

Virtual Memory control registers

Name	Type	Description
AMAIR_EL1	RW	*B2.15 AMAIR_EL1, Auxiliary Memory Attribute Indirection Register, EL1* on page B2-301
AMAIR_EL2	RW	*B2.16 AMAIR_EL2, Auxiliary Memory Attribute Indirection Register, EL2* on page B2-302
AMAIR_EL3	RW	*B2.17 AMAIR_EL3, Auxiliary Memory Attribute Indirection Register, EL3* on page B2-303
LORC_EL1	RW	*B2.76 LORC_EL1, LORegion Control Register, EL1* on page B2-396
LOREA_EL1	RW	LORegion End Address Register EL1
LORID_EL1	RO	*B2.78 LORID_EL1, Limited Order Region Identification Register, EL1* on page B2-398
LORN_EL1	RW	*B2.79 LORN_EL1, LORegion Number Register, EL1* on page B2-399
LORSA_EL1	RW	LORegion Start Address Register EL1
TCR_EL1	RW	*B2.90 TCR_EL1, Translation Control Register, EL1* on page B2-412
TCR_EL2	RW	*B2.91 TCR_EL2, Translation Control Register, EL2* on page B2-413
TCR_EL3	RW	*B2.92 TCR_EL3, Translation Control Register, EL3* on page B2-417
TTBR0_EL1	RW	*B2.93 TTBR0_EL1, Translation Table Base Register 0, EL1* on page B2-419
TTBR0_EL2	RW	*B2.94 TTBR0_EL2, Translation Table Base Register 0, EL2* on page B2-420
TTBR0_EL3	RW	*B2.95 TTBR0_EL3, Translation Table Base Register 0, EL3* on page B2-421
TTBR1_EL1	RW	*B2.96 TTBR1_EL1, Translation Table Base Register 1, EL1* on page B2-422
TTBR1_EL2	RW	*B2.97 TTBR1_EL2, Translation Table Base Register 1, EL2* on page B2-423
VTTBR_EL2	RW	*B2.103 VTTBR_EL2, Virtualization Translation Table Base Register, EL2* on page B2-432

Virtualization registers

Name	Type	Description
ACTLR_EL2	RW	*B2.6 ACTLR_EL2, Auxiliary Control Register, EL2* on page B2-290
AFSR0_EL2	RW	*B2.9 AFSR0_EL2, Auxiliary Fault Status Register 0, EL2* on page B2-295
AFSR1_EL2	RW	*B2.12 AFSR1_EL2, Auxiliary Fault Status Register 1, EL2* on page B2-298
AMAIR_EL2	RW	*B2.16 AMAIR_EL2, Auxiliary Memory Attribute Indirection Register, EL2* on page B2-302
CPTR_EL2	RW	*B2.21 CPTR_EL2, Architectural Feature Trap Register, EL2* on page B2-309
ESR_EL2	RW	*B2.47 ESR_EL2, Exception Syndrome Register, EL2* on page B2-348
HACR_EL2	RW	*B2.49 HACR_EL2, Hyp Auxiliary Configuration Register, EL2* on page B2-350
HCR_EL2	RW	*B2.50 HCR_EL2, Hypervisor Configuration Register, EL2* on page B2-351
HPFAR_EL2	RW	Hypervisor IPA Fault Address Register EL2
TCR_EL2	RW	*B2.91 TCR_EL2, Translation Control Register, EL2* on page B2-413
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 B2-429
VTCR_EL2	RW	*B2.102 VTCR_EL2, Virtualization Translation Control Register, EL2* on page B2-431
VTTBR_EL2	RW	*B2.103 VTTBR_EL2, Virtualization Translation Table Base Register, EL2* on page B2-432

Exception and fault handling registers

Name	Type	Description
AFSR0_EL1	RW	*B2.8 AFSR0_EL1, Auxiliary Fault Status Register 0, EL1* on page B2-294
AFSR0_EL2	RW	*B2.9 AFSR0_EL2, Auxiliary Fault Status Register 0, EL2* on page B2-295
AFSR0_EL3	RW	*B2.10 AFSR0_EL3, Auxiliary Fault Status Register 0, EL3* on page B2-296
AFSR1_EL1	RW	*B2.11 AFSR1_EL1, Auxiliary Fault Status Register 1, EL1* on page B2-297
AFSR1_EL2	RW	*B2.12 AFSR1_EL2, Auxiliary Fault Status Register 1, EL2* on page B2-298
AFSR1_EL3	RW	*B2.13 AFSR1_EL3, Auxiliary Fault Status Register 1, EL3* on page B2-299
DISR_EL1	RW	*B2.34 DISR_EL1, Deferred Interrupt Status Register, EL1* on page B2-332
ESR_EL1	RW	*B2.46 ESR_EL1, Exception Syndrome Register, EL1* on page B2-347
ESR_EL2	RW	*B2.47 ESR_EL2, Exception Syndrome Register, EL2* on page B2-348
ESR_EL3	RW	*B2.48 ESR_EL3, Exception Syndrome Register, EL3* on page B2-349
HPFAR_EL2	RW	Hypervisor IPA Fault Address Register EL2
IFSR32_EL2	RW	*B2.75 IFSR32_EL2, Instruction Fault Status Register, EL2* on page B2-394
VDISR_EL2	RW	*B2.98 VDISR_EL2, Virtual Deferred Interrupt Status Register, EL2* on page B2-424
VSESR_EL2	RW	*B2.101 VSESR_EL2, Virtual SError Exception Syndrome Register* on page B2-429

Implementation defined registers

Name	Type	Description
CPUACTLR_EL1	RW	*B2.23 CPUACTLR_EL1, CPU Auxiliary Control Register, EL1* on page B2-311
CPUCFR_EL1	RO	*B2.24 CPUCFR_EL1, CPU Configuration Register, EL1* on page B2-313
CPUECTLR_EL1	RW	*B2.25 CPUECTLR_EL1, CPU Extended Control Register, EL1* on page B2-315
CPUPCR_EL3	RW	*B2.26 CPUPCR_EL3, CPU Private Control Register, EL3* on page B2-318
CPUPMR_EL3	RW	*B2.27 CPUPMR_EL3, CPU Private Mask Register, EL3* on page B2-320
CPUPOR_EL3	RW	*B2.28 CPUPOR_EL3, CPU Private Operation Register, EL3* on page B2-322
CPUPSELR_EL3	RW	*B2.29 CPUPSELR_EL3, CPU Private Selection Register, EL3* on page B2-324
CPUPWRCTLR_EL1	RW	*B2.30 CPUPWRCTLR_EL1, Power Control Register, EL1* on page B2-326
ERXPFGCDNR_EL1	RW	*B2.42 ERXPFGCDNR_EL1, Selected Error Pseudo Fault Generation Count Down Register, EL1* on page B2-341

(continued)

Name	Type	Description
ERXPFGCTLR_EL1	RW	*B2.43 ERXPFGCTLR_EL1, Selected Error Pseudo Fault Generation Control Register, EL1* on page B2-343
ERXPFGFR_EL1	RW	*B2.44 ERXPFGFR_EL1, Selected Pseudo Fault Generation Feature Register, EL1* on page B2-345

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 or 64-bit	Cluster PMU registers

Security

Name	Type	Description
ACTLR_EL3	RW	*B2.7 ACTLR_EL3, Auxiliary Control Register, EL3* on page B2-292
AFSR0_EL3	RW	*B2.10 AFSR0_EL3, Auxiliary Fault Status Register 0, EL3* on page B2-296
AFSR1_EL3	RW	*B2.13 AFSR1_EL3, Auxiliary Fault Status Register 1, EL3* on page B2-299
AMAIR_EL3	RW	*B2.17 AMAIR_EL3, Auxiliary Memory Attribute Indirection Register, EL3* on page B2-303
CPTR_EL3	RW	*B2.22 CPTR_EL3, Architectural Feature Trap Register, EL3* on page B2-310
MDCR_EL3	RW	*B2.81 MDCR_EL3, Monitor Debug Configuration Register, EL3* on page B2-401

Reset management registers

Name	Type	Description
RVBAR_EL3	RW	*B2.86 RVBAR_EL3, Reset Vector Base Address Register, EL3* on page B2-408

Address registers

Name	Type	Description
PAR_EL1	RW	*B2.84 PAR_EL1, Physical Address Register, EL1* on page B2-406

B2.5 ACTLR_EL1, Auxiliary Control Register, EL1

ACTLR_EL1 provides IMPLEMENTATION DEFINED configuration and control options for execution at EL1 and EL0.

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

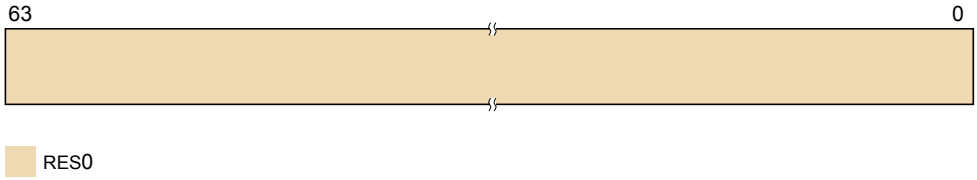


Figure B2-1 ACTLR_EL1 bit assignments

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 B1-128.

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 B1-130.

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.

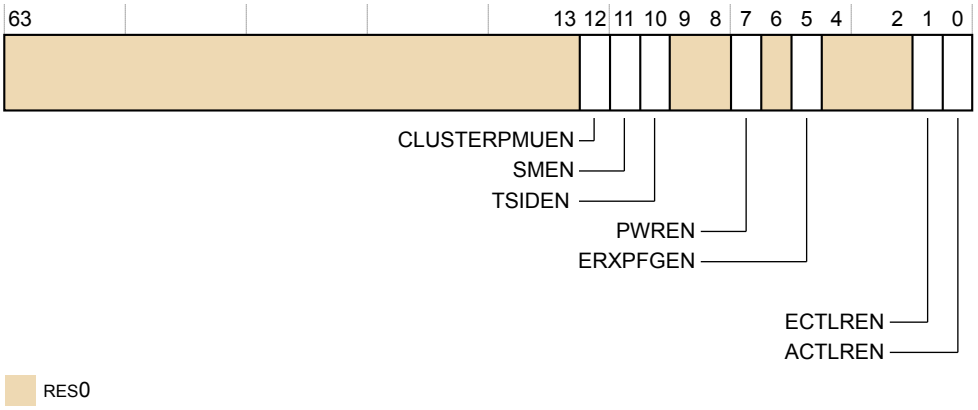


Figure B2-2 ACTLR_EL2 bit assignments

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.

RES0, [4:2]

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 and CLUSTERACTLR are not write-accessible from EL1 Non-secure. This is the reset value.

1 CPUACTLR 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 *B1.45 HACTLR, Hyp Auxiliary Control Register* on page B1-187.

ACTLR_EL2 bits [63:32] are architecturally mapped to the AArch32 HACTLR2 register. See *B1.46 HACTLR2, Hyp Auxiliary Control Register 2* on page B1-189.

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.

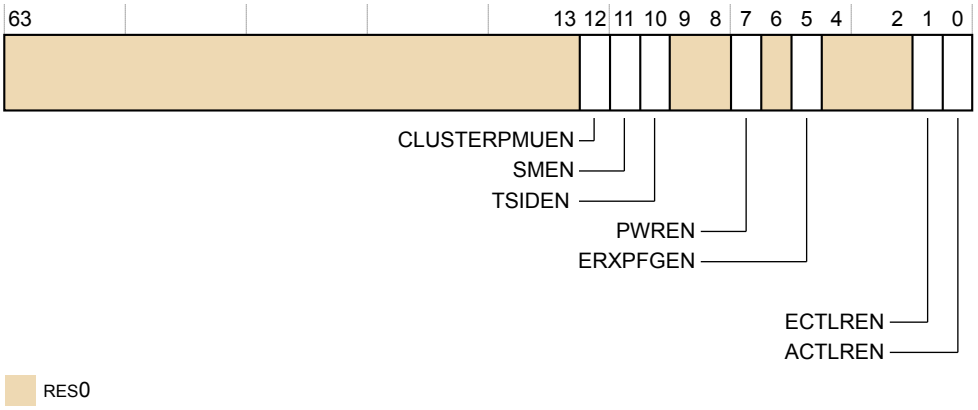


Figure B2-3 ACTLR_EL3 bit assignments

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.

RES0, [4:2]

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 and CLUSTERACTLR are not write-accessible from EL2 and EL1 Secure. This is the reset value.

`1` CPUACTLR 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 B1-128.

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 B1-130.

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-A55 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.

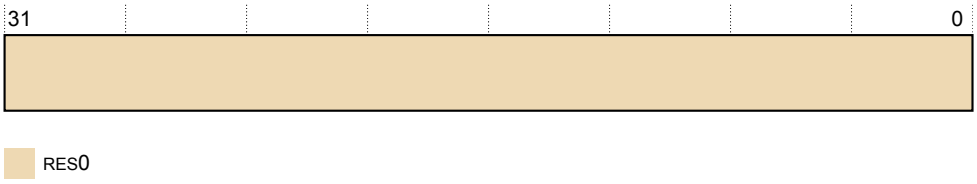


Figure B2-4 AFSR0_EL1 bit assignments

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 B1-131.

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.

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.

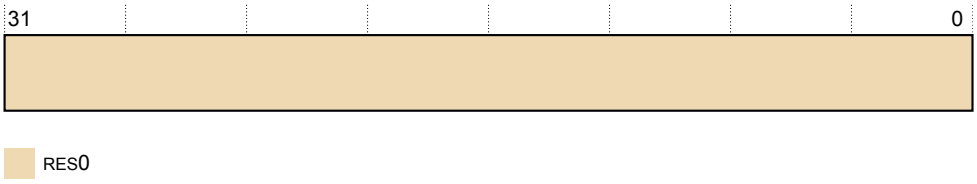


Figure B2-5 AFSR0_EL2 bit assignments

RES0, [31:0]

Reserved, RES0.

Configurations

AArch64 System register AFSR0_EL2 is architecturally mapped to AArch32 System register HADFSR. See *B1.47 HADFSR, Hyp Auxiliary Data Fault Status Syndrome Register* on page B1-190.

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-A55 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.

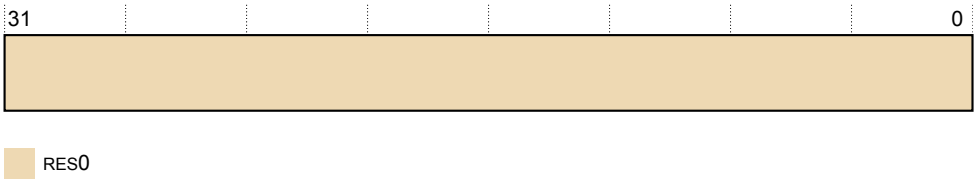


Figure B2-6 AFSR0_EL3 bit assignments

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-A55.

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.

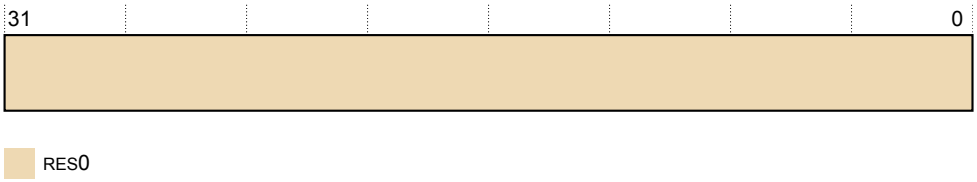


Figure B2-7 AFSR1_EL1 bit assignments

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 B1-133.

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-A55 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.

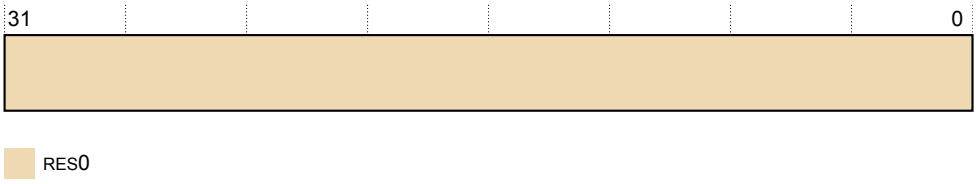


Figure B2-8 AFSR1_EL2 bit assignments

RES0, [31:0]

Reserved, RES0.

Configurations

AArch64 System register AFSR1_EL2 is architecturally mapped to AArch32 System register HAIFSR. See *B1.48 HAIFSR, Hyp Auxiliary Instruction Fault Status Syndrome Register* on page B1-191.

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-A55 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.

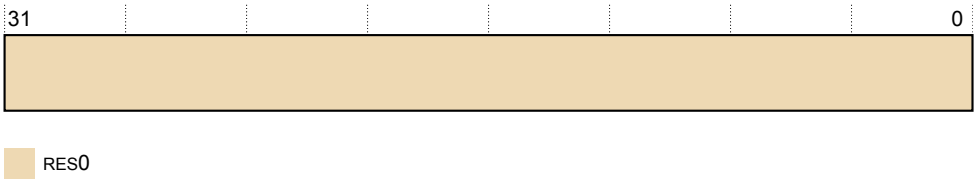


Figure B2-9 AFSR1_EL3 bit assignments

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 Cortex-A55 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.

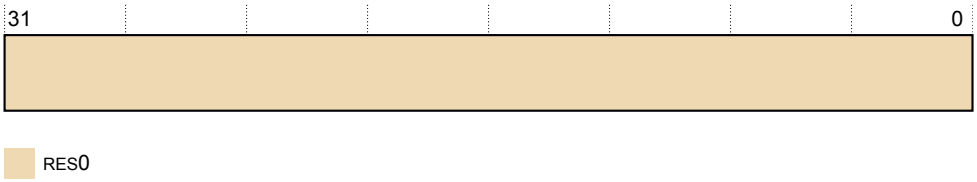


Figure B2-10 AIDR_EL1 bit assignments

RES0, [31:0]

Reserved, RES0.

Configurations

AIDR_EL1 is architecturally mapped to AArch32 register AIDR. See *B1.8 AIDR, Auxiliary ID Register* on page B1-132.

Bit fields and details not provided in this description are architecturally defined. See the *ARM® Architecture Reference Manual ARMv8, for ARMv8-A architecture profile*.

B2.15 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-A55 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.

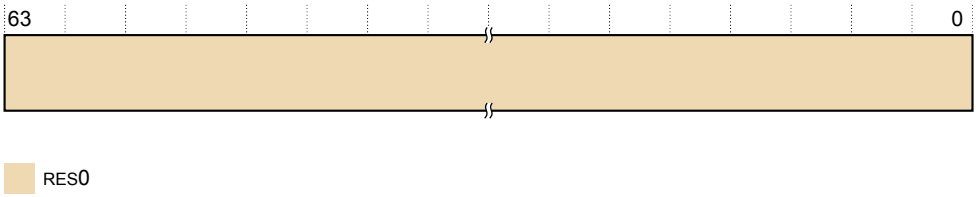


Figure B2-11 AMAIR_EL1 bit assignments

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 B1-134.

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 B1-135.

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-A55 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.

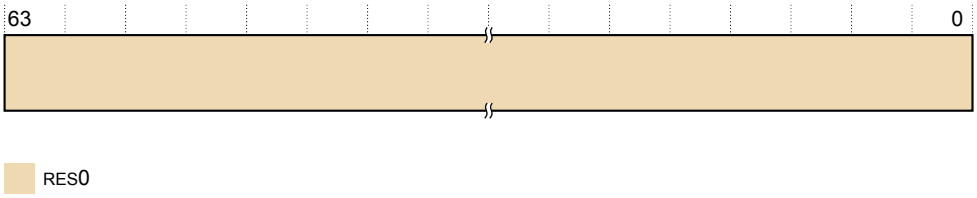


Figure B2-12 AMAIR_EL1 bit assignments

RES0, [63:0]

Reserved, RES0.

Configurations

AArch64 System register AMAIR_EL2 bits [31:0] are architecturally mapped to AArch32 System register HAMAIR0. See *B1.49 HAMAIR0, Hyp Auxiliary Memory Attribute Indirection Register 0* on page B1-192.

AArch64 System register AMAIR_EL2 bits [63:32] are architecturally mapped to AArch32 System register HAMAIR1. See *B1.50 HAMAIR1, Hyp Auxiliary Memory Attribute Indirection Register 1* on page B1-193.

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-A55 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.

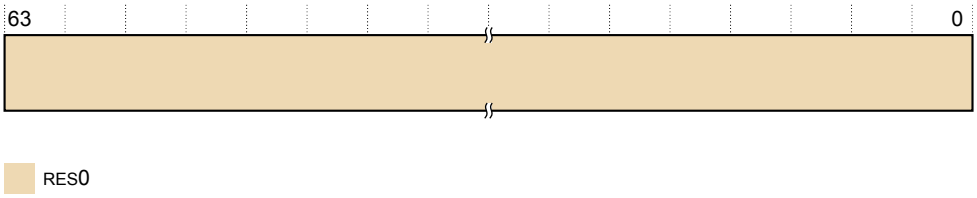


Figure B2-13 AMAIR_EL3 bit assignments

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.

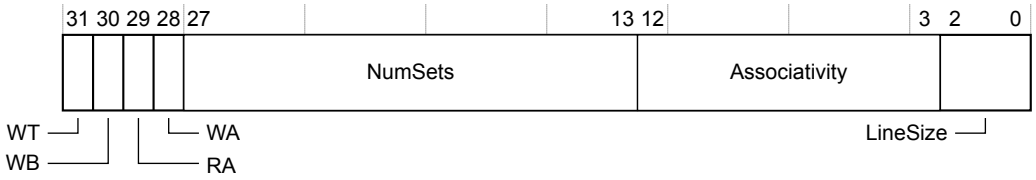


Figure B2-14 CCSIDR_EL1 bit assignments

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 B2-305.

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 B2-305.

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 B2-305.

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 B2-305.

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 B2-305.

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 B2-305.

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 B2-305.

Configurations

CCSIDR_EL1 is architecturally mapped to AArch32 register CCSIDR. See *B1.12 CCSIDR, Cache Size ID Register* on page B1-136.

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	16KB	7007E01A	0	1	1	1	003F	003	2
			32KB	700FE01A					007F	003	2
			64KB	701FE01A					00FF	003	2
0b000	0b1	L1 Instruction cache	16KB	2007E01A	0	0	1	0	003F	003	2
			32KB	200FE01A					007F	003	2
			64KB	201FE01A					00FF	003	2
0b001	0b0	L2 cache	Not present	See following Note.	-	-	-	-	-	-	-
			64KB	701FE01A	0	1	1	1	00FF	003	2
			128KB	703FE01A					01FF	003	2
			256KB	707FE01A					03FF	003	2
0b001	0b1	Reserved	-	-	-	-	-	-	-	-	-
0b010	0b0	L3 cache	512KB	703FE07A	0	1	1	1	01FF	00F	2
			1024KB	707FE07A					03FF	00F	2
			2048KB	70FFE07A					07FF	00F	2
			4096KB	71FFE07A					0FFF	00F	2
0b0101 - 0b1111		Reserved	-	-	-	-	-	-	-	-	-

Note

If no L2 cache is present the core uses L3 cache as L2, and the L3 encodings apply.

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.

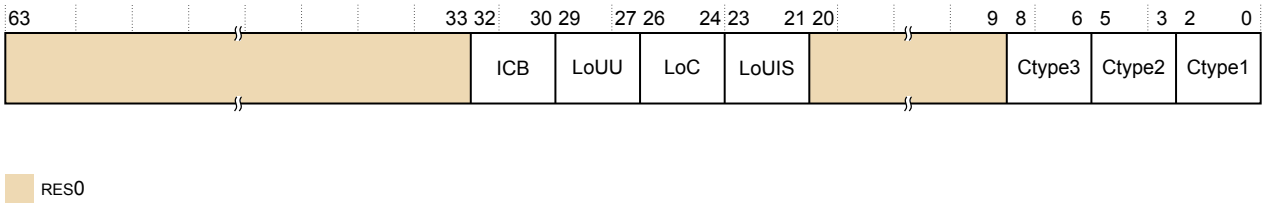


Figure B2-15 CLIDR_EL1 bit assignments

RES0, [63:33]

RES0 Reserved.

ICB, [32:30]

Inner cache boundary. This field indicates the boundary between the inner and the outer domain:

001 L1 cache is the highest inner level.

010 L2 cache is the highest inner level.

011 L3 cache is the highest inner level.

LoUU, [29:27]

Indicates the Level of Unification Uniprocessor for the cache hierarchy:

000 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:

001 L2 and L3 cache are not implemented.

010 L2 or L3 cache is not implemented.

011 L2 and L3 cache are implemented.

LoUIS, [23:21]

Indicates the *Level of Unification Inner Shareable* (LoUIS) for the cache hierarchy.

0b000 No levels of cache need to be cleaned or invalidated when cleaning or invalidating to the 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 cluster implements L3 cache. If present, unified instruction and data caches at Level-3:

`000` L2 or L3 cache is not implemented.

`100` L2 and L3 cache are implemented.

If Ctype2 has a value of `3b000`, the value of Ctype3 must be ignored.

Ctype2, [5:3]

Indicates the type of cache if the core implements L2 cache. If present, unified instruction and data caches at Level-2:

`000` L2 and L3 cache are not implemented.

`100` L2 or L3 cache is implemented as a unified cache.

Ctype1, [2:0]

Indicates the type of cache implemented at L1:

`011` 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 B1-139.

B2.20 CPACR_EL1, Architectural Feature Access Control Register, EL1

The CPACR_EL1 controls access to floating-point, and Advanced SIMD functionality from EL0, EL1, and EL3.

Bit field descriptions

CPACR_EL1 is a 32-bit register, and is part of the Other system control registers functional group.

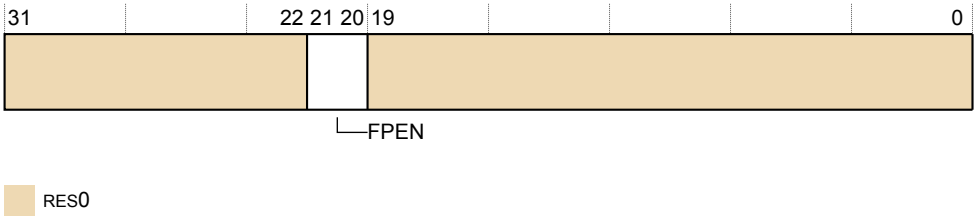


Figure B2-16 CPACR_EL1 bit assignments

RES0, [31:22]

RES0 Reserved.

FPEN, [21:20]

Traps instructions that access registers associated with Advanced SIMD and floating-point execution to trap to EL1 when executed from EL0 or EL1. The possible values are:

`0b00` Trap any instruction in EL0 or EL1 that uses registers associated with Advanced SIMD and floating-point execution. The reset value is `0b00`.

`0b01` Trap any instruction in EL0 that uses registers associated with Advanced SIMD and floating-point execution. Instructions in EL1 are not trapped.

`0b10` Trap any instruction in EL0 or EL1 that uses registers associated with Advanced SIMD and floating-point execution.

`0b11` No instructions are trapped.

This field is RES0 if Advanced SIMD and floating-point are not implemented.

RES0, [19:0]

RES0 Reserved.

Configurations

CPACR_EL1 is architecturally mapped to AArch32 register CPACR. See *B1.14 CPACR, Architectural Feature Access Control Register* on page B1-141.

Bit fields and details not provided in this description are architecturally defined. See the *ARM® Architecture Reference Manual ARMv8, for ARMv8-A architecture profile*.

B2.21 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.

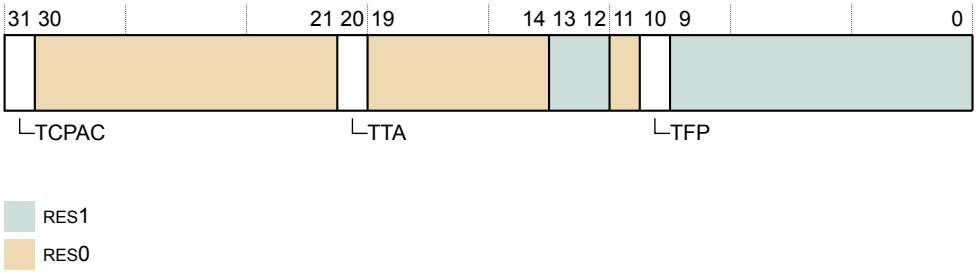


Figure B2-17 CPTR_EL2 bit assignments

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.

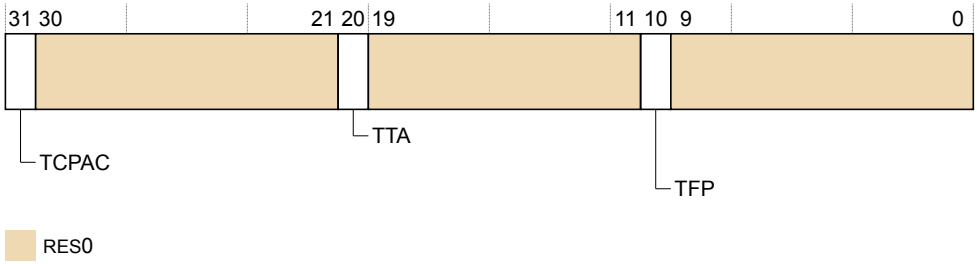


Figure B2-18 CPTR_EL3 bit assignments

TTA, [20]

Trap Trace Access.

Not implemented. RES0.

TFP, [10]

This causes instructions that access the registers that are associated with Advanced SIMD or floating-point execution to trap to EL3 when executed from any Exception level, unless trapped to EL1 or EL2. The possible values are: are:

`0` Does not cause any instruction to be trapped. This is the reset value if the Advanced SIMD and floating-point support is implemented.

`1` Causes any instructions that use the registers that are associated with Advanced SIMD or floating-point execution to be trapped. This is always the value if the Advanced SIMD and floating-point support is not implemented.

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.

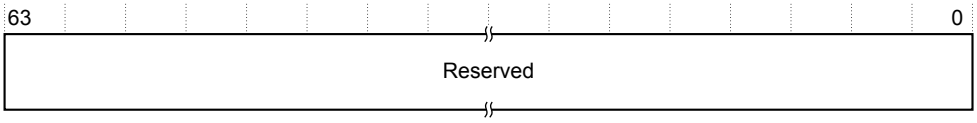


Figure B2-19 CPUACTLR_EL1 bit assignments

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 B1-142.

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

'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 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.

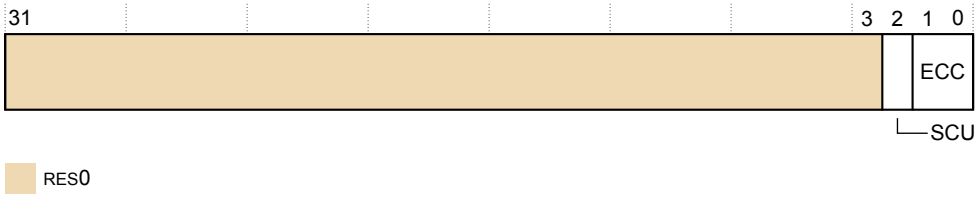


Figure B2-20 CPUCFR_EL1 bit assignments

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 *B1.16 CPUCFR, CPU Configuration 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*.

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.

B2.25 CPUECTLR_EL1, CPU Extended Control Register, EL1

The CPUECTLR_EL1 provides extra 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.

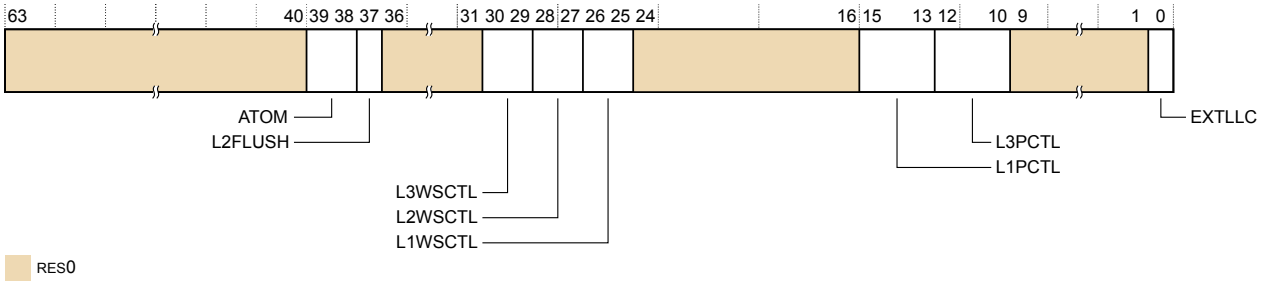


Figure B2-21 CPUECTLR_EL1 bit assignments

RES0, [63:40]

RES0 Reserved.

ATOM, [39:38]

00 Atomic instructions are performed near if they hit in the cache in a unique state, or far if they miss or are shared. For more details, see *A6.4.1 Memory system implementation* on page A6-78. This is the default.

01 Force all cacheable atomic instructions to be executed near, in the L1 cache.

10 Force most cacheable atomic instructions to be executed far, in the L3 cache or beyond.

11 Force cacheable load atomics, including SWP and CAS, to be executed near, in the L1 cache. Store atomics are performed near if they hit in the cache in a unique state, or far if they miss or are shared.

L2FLUSH, [37]

0 L2 cache flushes, for example during a core powerdown sequence, cause clean lines to be allocated into the L3 cache rather than discarding them. This can improve performance if it is known that the data is likely to be used soon by another core.

RES0, [36:31]

RES0 Reserved.

L3WSCTL, [30:29]

Write streaming no-L3-allocate threshold. The possible values are:

00 128th consecutive streaming cache line does not allocate in the L1, L2, or L3 cache.

01 1024th consecutive streaming cache line does not allocate in the L1, L2, or L3 cache. This is the reset value.

10 4096th consecutive streaming cache line does not allocate in the L1, L2, or L3 cache.

11 Disables streaming. All write-allocate lines allocate in the L1, L2, or L3 cache.

L2WSCTL, [28:27]

Write streaming no-L2-allocate threshold. The possible values are:

00 16th consecutive streaming cache line does not allocate in the L1 or L2 cache.

01 128th consecutive streaming cache line does not allocate in the L1 or L2 cache. This is the reset value.

10 512th consecutive streaming cache line does not allocate in the L1 or L2 cache.

11 Disables streaming. All write-allocate lines allocate in the L1 or L2 cache.

L1WSCTL, [26:25]

Write streaming no-L1-allocate threshold. The possible values are:

00 4th consecutive streaming cache line does not allocate in the L1 cache. This is the reset value.

01 64th consecutive streaming cache line does not allocate in the L1 cache.

10 128th consecutive streaming cache line does not allocate in the L1

11 Disables streaming. All write-allocate lines allocate in the L1 cache.

RES0, [24:16]

RES0 Reserved.

L1PCTL, [15:13]

L1 Data prefetch control. The value of the L1PCTL field determines the maximum number of outstanding data prefetches allowed in the L1 memory system (not counting the data prefetches generated by software load/PLD instructions).

000 Prefetch disabled.

001 1 outstanding prefetch allowed.

010 2 outstanding prefetches allowed.

011 3 outstanding prefetches allowed.

100 4 outstanding prefetches allowed.

101 5 outstanding prefetches allowed. This is the reset value.

110 6 outstanding prefetches allowed.

111 7 outstanding prefetches allowed.

L3PCTL, [12:10]

L3 Data prefetch control. The value of the L3PCTL field determines the approximate distance between the L1 prefetcher and requests sent to the L3 memory system. Increasing this distance may improve performance on systems with higher latency to main memory, but increasing it too far can reduce performance.

——— **Note** ———

The L3 memory system may have more outstanding access to the system than this number.

———————————

000 Fetch 16 lines ahead.

001 Fetch 32 lines ahead.

010 Reserved.

011 Reserved.

100 Disable L3 prefetching.

101 Fetch 2 lines ahead.

110 Fetch 4 lines ahead.

111 Fetch 8 lines ahead. This is the reset value.

RES0, [9:1]

RES0 Reserved.

EXTLLC, [0]

`0` 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_EL1 is mapped to the AArch32 CPUECTLR register. See *B1.17 CPUECTLR, CPU Extended Control Register* on page B1-146.

Usage constraints

Accessing the CPUECTLR_EL1

The CPUECTLR_EL1 can be written dynamically.

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 MRS 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
CPUECTLR_EL1	11	000	1111	0001	100

Accessibility

This register is accessible in software as follows:

<systemreg>	Control			Accessibility			
	E2H	TGE	NS	EL0	EL1	EL2	EL3
CPUECTLR_EL1	x	x	0	-	RW	n/a	RW
CPUECTLR_EL1	x	0	1	-	RW	RW	RW
CPUECTLR_EL1	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 *Synchronous exception prioritization* in the *ARM® Architecture Reference Manual ARMv8, for ARMv8-A architecture profile*.

B2.26 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.

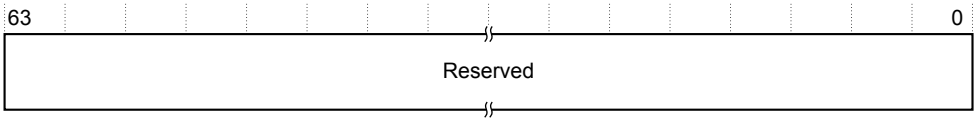


Figure B2-22 CPUPCR_EL3 bit assignments

Reserved, [63:0]

Reserved for ARM internal use.

Configurations

CPUPCR_EL3 is:

- Only accessible in Secure state.
- Mapped to the AArch32 CPUPCR register. See *B1.18 CPUPCR, CPU Private Control Register* on page B1-149.

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*.

B2.27 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.

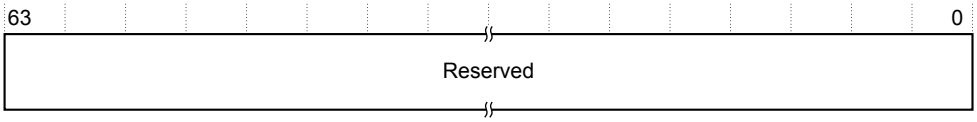


Figure B2-23 CPUPMR_EL3 bit assignments

Reserved, [63:0]

Reserved for ARM internal use.

Configurations

CPUPMR_EL3 is:

- Only accessible from Secure state.
- Mapped to the AArch32 CPUPMR register. See *B1.19 CPUPMR, CPU Private Mask Register* on page B1-151.

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*.

B2.28 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.

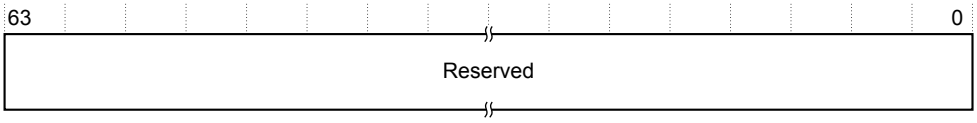


Figure B2-24 CPUPOR_EL3 bit assignments

Reserved, [63:0]

Reserved for ARM internal use.

Configurations

CPUPOR_EL3 is:

- Only accessible in Secure state.
- Mapped to the AArch32 CPUPOR register. See *B1.20 CPUPOR, CPU Private Operation Register* on page B1-153.

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*.

B2.29 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.

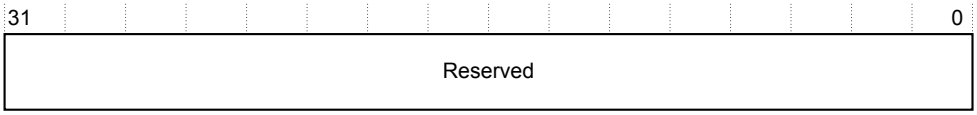


Figure B2-25 CPUPSELR_EL3 bit assignments

Reserved, [31:0]

Reserved for ARM internal use.

Configurations

CPUPSELR_EL3 is:

- Only accessible in Secure state.
- Mapped to the AArch32 CPUPSELR register. See *B1.21 CPUPSELR, CPU Private Selection Register* on page B1-155.

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*.

B2.30 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.

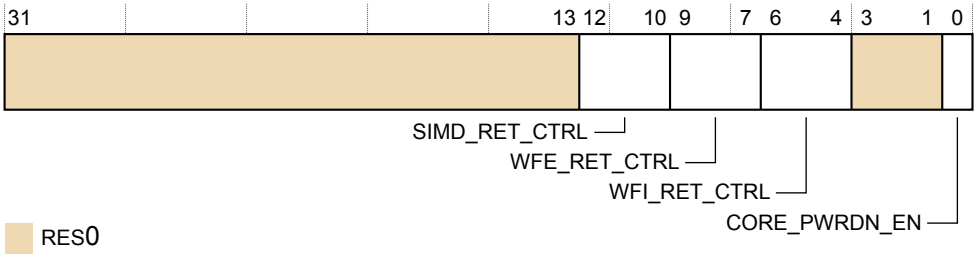


Figure B2-26 CPUPWRCTLR_EL1 bit assignments

RES0, [31:13]

RES0 Reserved.

SIMD_RET_CTRL, [12:10]

Advanced SIMD and floating-point retention control:

000 Disable the retention circuit. This is the default value, see *Table B2-7 CPUPWRCTLR Retention Control Field* on page B2-327 for more retention control options.

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 B2-327 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 B2-327 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
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 *B1.22 CPUPWRCTLR, CPU Power Control Register* on page B1-157.

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.

B2.31 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 B2-304.

Bit field descriptions

CSSELR_EL1 is a 32-bit register, and is part of the Identification registers functional group.

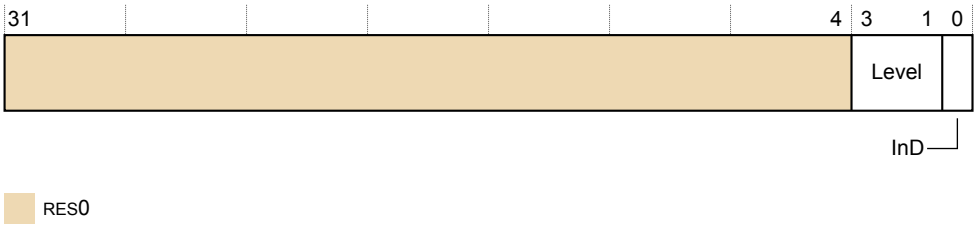


Figure B2-27 CSSELR_EL1 bit assignments

[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 *B1.23 CSSELR, Cache Size Selection Register* on page B1-159.

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*.

B2.32 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.

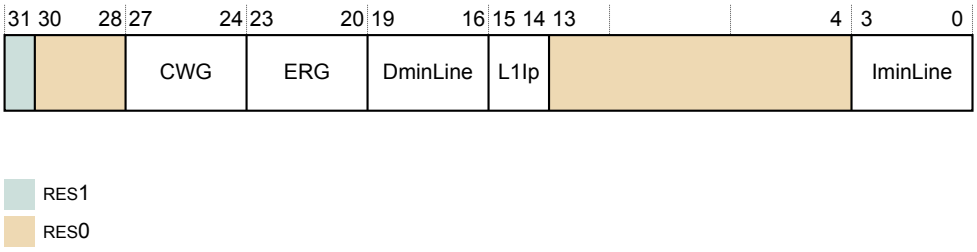


Figure B2-28 CTR_EL0 bit assignments

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 *B1.24 CTR, Cache Type Register* on page B1-160.

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*.

B2.33 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.

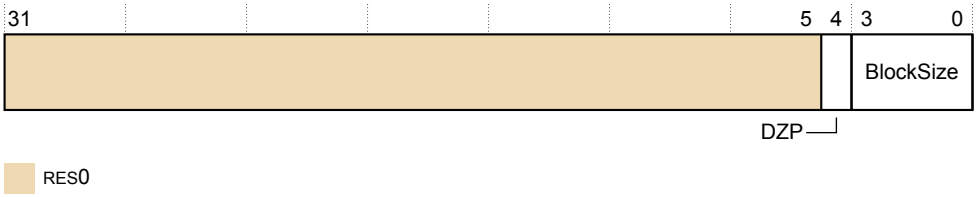


Figure B2-29 DCZID_EL0 bit assignments

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*.

B2.34 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.

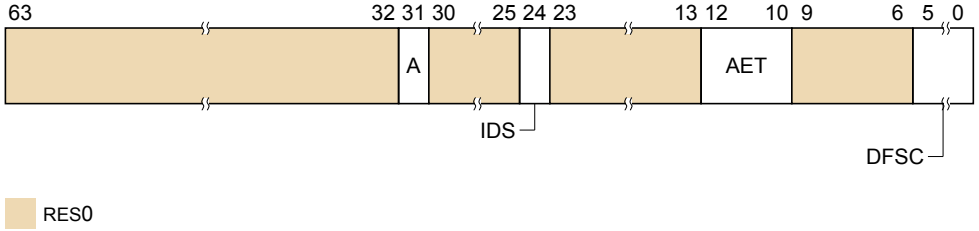


Figure B2-30 DISR_EL1 bit assignments, DISR_EL1.IDS is 0

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 value or values are:

`0b001` Unrecoverable error (UEU).

All other values are reserved. Reserved values might be defined in a future version of the architecture.

In the event of multiple errors taken as a single SError interrupt exception, the overall state of the PE is reported. For example, if both a Recoverable and Unrecoverable error occurred, the state is Unrecoverable.

Note

- This field is only valid if IDS == `0b0` and DFSC == `0b010001`.
- 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. The possible values are:

`0b010001` Asynchronous SError Interrupt when the core is executing in AArch64 state or at EL2, or when the core is executing in AArch32 state at EL1, and the Extended Address Enable bit is at 1 in TTBCR.

`0b000110` Asynchronous SError Interrupt when the core is executing in AArch32 state at EL1 and the Extended Address Enable is at 0 in TTBCR.

Configurations

AArch64 System register DISR_EL1 is architecturally mapped to AArch32 register DISR. See *B1.26 DISR, Deferred Interrupt Status Register* on page B1-164.

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 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.

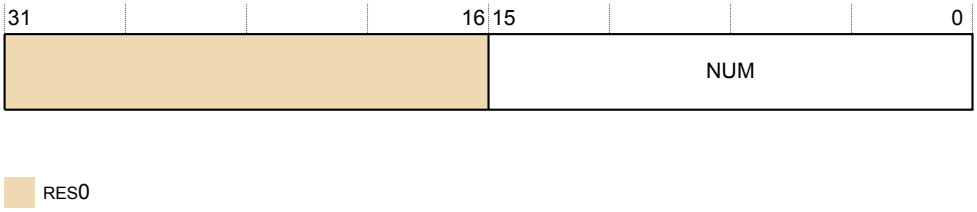


Figure B2-31 ERRIDR_EL1 bit assignments

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 *B1.27 ERRIDR, Error ID Register* on page B1-167.

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 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.

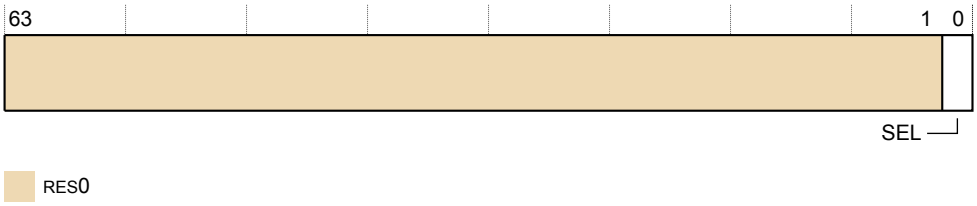


Figure B2-32 ERRSELR_EL1 bit assignments

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-A55 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 *B1.28 ERRSELR, Error Record Select Register* on page B1-168.

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 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 B3-436.

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*.

B2.38 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 B3-437.

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*.

B2.39 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 B3-439.

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*.

B2.40 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 B3-441.

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*.

B2.41 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 B3-443.

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*.

B2.42 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 B3-444.

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 *B1.39 ERXPFGCDNR, Selected Error Pseudo Fault Generation Count Down Register* on page B1-179.

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 B2-290 and *B2.7 ACTLR_EL3, Auxiliary Control Register, EL3* on page B2-292.

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.

B2.43 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 B3-445.

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 *B1.40 ERXPFGCTLR, Selected Error Pseudo Fault Generation Control Register* on page B1-181.

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 B2-290 and *B2.7 ACTLR_EL3, Auxiliary Control Register, EL3* on page B2-292.

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.

B2.44 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 B3-447.

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 *B1.41 ERXPFGFR, Selected Pseudo Fault Generation Feature Register* on page B1-183.

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 B2-290 and *B2.7 ACTLR_EL3, Auxiliary Control Register, EL3* on page B2-292.

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.

B2.45 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 B3-449.

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*.

B2.46 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.

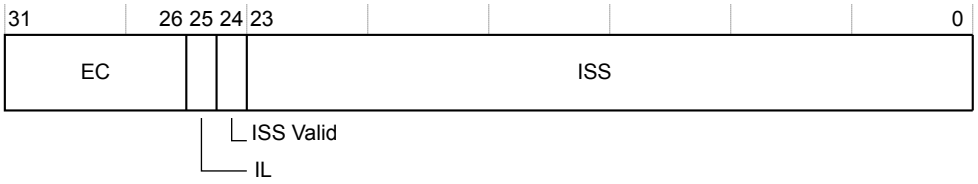


Figure B2-33 ESR_EL1 bit assignments

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 Valid, [24]
Syndrome valid. The possible values are:

`0` ISS not valid, ISS is RES0.

`1` ISS valid.

ISS, [23:0]
Syndrome information.

When the EC field is `0x2F`, indicating an SError interrupt has occurred, the ISS field contents are IMPLEMENTATION DEFINED.

Configurations
ESR_EL1 is architecturally mapped to AArch32 register DFSR (NS). See *B1.25 DFSR, Data Fault Status Register* on page B1-162.

Bit fields and details not provided in this description are architecturally defined. See the *ARM® Architecture Reference Manual ARMv8, for ARMv8-A architecture profile*.

B2.47 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.

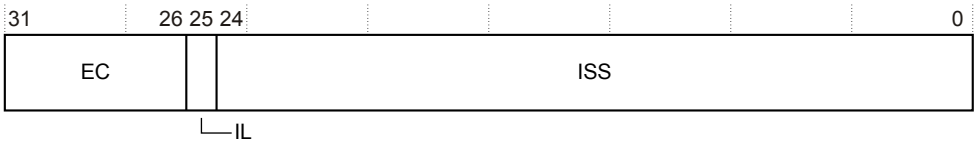


Figure B2-34 ESR_EL2 bit assignments

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 B2-429.

Configurations
ESR_EL2 is architecturally mapped to AArch32 register HSR. See *B1.54 HSR, Hyp Syndrome Register* on page B1-199.

If EL2 is not implemented, this register is RES0 from EL3.

RW fields in this register reset to architecturally UNKNOWN values.

B2.48 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.

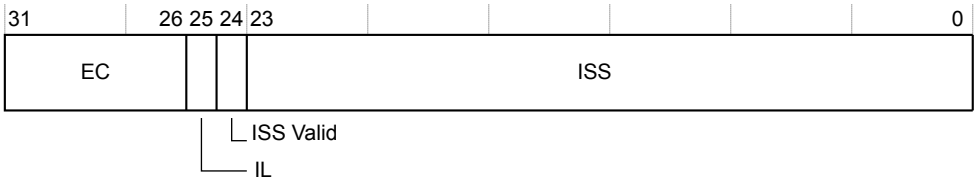


Figure B2-35 ESR_EL3 bit assignments

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 Valid, [24]
Syndrome valid. The possible values are:

`0` ISS not valid, ISS is RES0.

`1` ISS valid.

ISS, [23:0]
Syndrome information.

When the EC field is `0x2F`, indicating an SError interrupt has occurred, the ISS field contents are IMPLEMENTATION DEFINED.

Configurations
ESR_EL3 is mapped to AArch32 register DFSR(S). See *B1.25 DFSR, Data Fault Status Register* on page B1-162.

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*.

B2.49 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-A55 core.

Bit field descriptions

HACR_EL2 is a 32-bit register, and is part of Virtualization registers functional group.

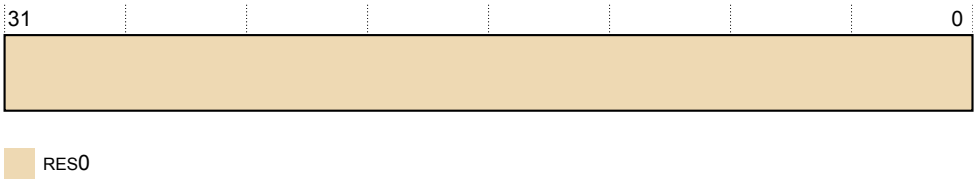


Figure B2-36 HACR_EL2 bit assignments

RES0, [31:0]

Reserved, RES0.

Configurations

AArch64 System register HACR_EL2 is architecturally mapped to AArch32 System register HACR. See *B1.44 HACR, Hyp Auxiliary Configuration Register* on page B1-186.

Bit fields and details not provided in this description are architecturally defined. See the *ARM® Architecture Reference Manual ARMv8, for ARMv8-A architecture profile*.

B2.50 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.

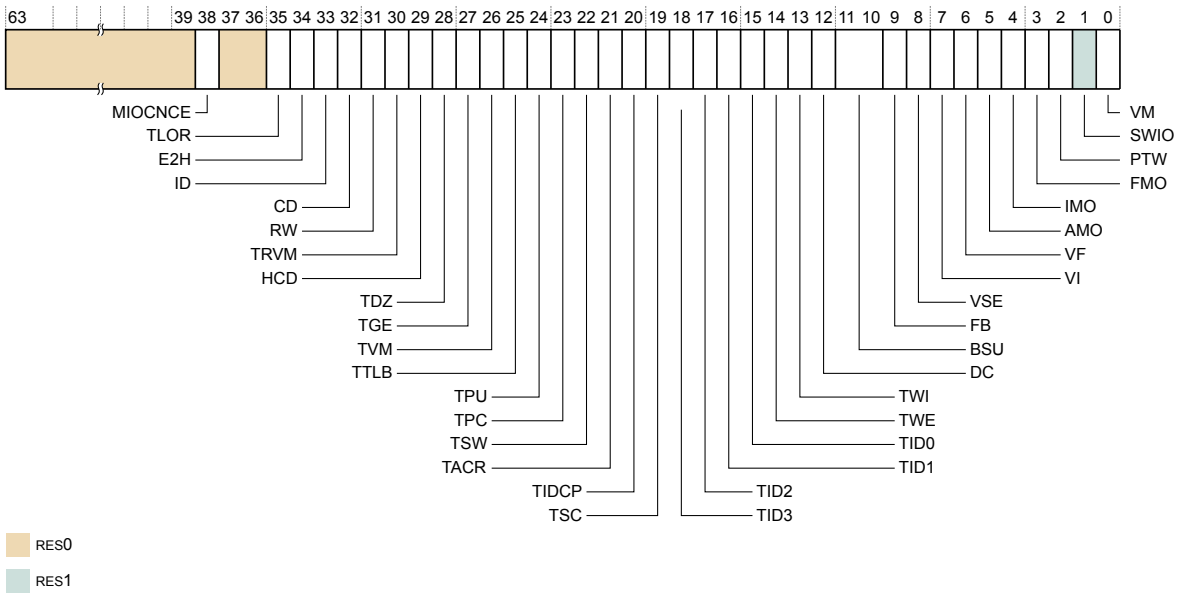


Figure B2-37 HCR_EL2 bit assignments

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 *B1.51 HCR, Hyp Configuration Register* on page B1-194.

HCR_EL2[63:32] is architecturally mapped to AArch32 register HCR2. See *B1.52 HCR2, Hyp Configuration Register 2* on page B1-196.

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*.

B2.51 HPFAR_EL2, Hypervisor IPA Fault Address Register, EL2

The HPFAR_EL2 holds the faulting IPA for some aborts on a stage 2 translation taken to EL2.

Bit field descriptions

HPFAR_EL2 is a 64-bit register, and is part of:

- The Exception and fault handling registers functional group.
- The Virtualization registers functional group.

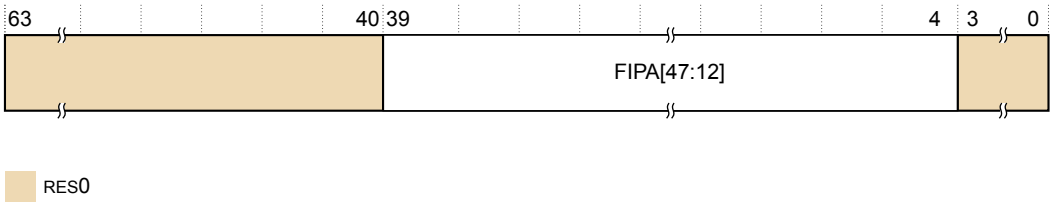


Figure B2-38 HPFAR_EL2 bit assignments

RES0, [63:40]

RES0 Reserved.

FIPA[47:12], [39:4]

Bits [47:12] of the faulting intermediate physical address. The equivalent upper bits in this field are RES0.

RES0, [3:0]

RES0 Reserved.

Configurations

AArch64 register HPFAR_EL2[31:0] is architecturally mapped to AArch32 register HPFAR. See the *ARM® Architecture Reference Manual ARMv8, for ARMv8-A architecture profile*.

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_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.

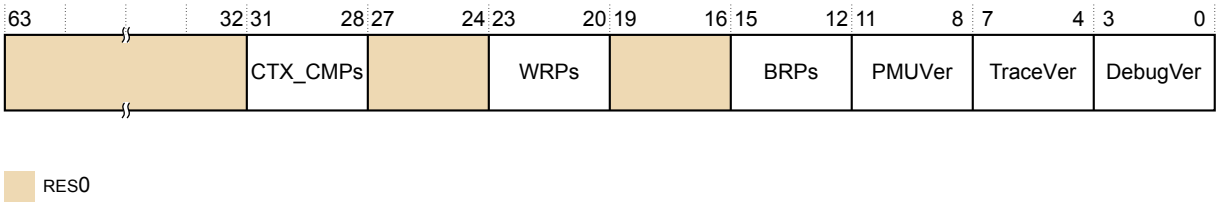


Figure B2-39 ID_AA64DFR0_EL1 bit assignments

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*.

B2.53 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.

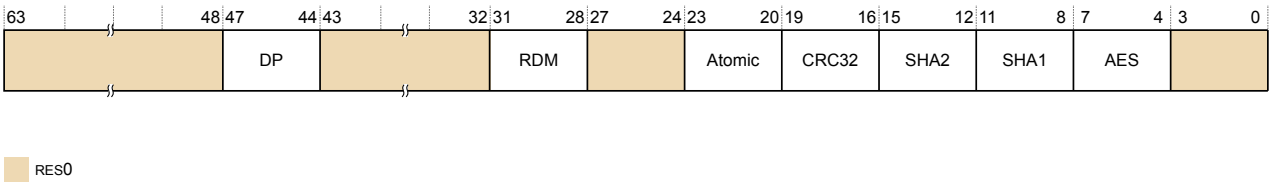


Figure B2-40 ID_AA64ISAR0_EL1 bit assignments

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*.

B2.54 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.

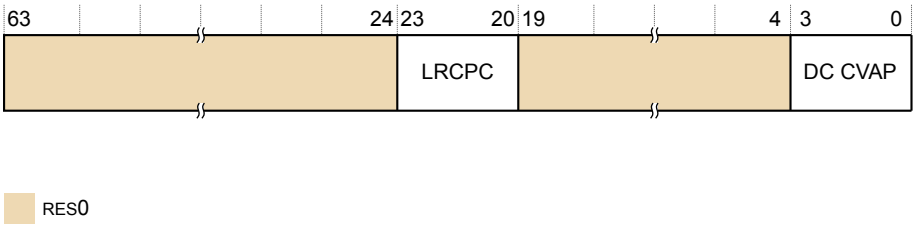


Figure B2-41 ID_AA64ISAR1_EL1 bit assignments

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*.

B2.55 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.

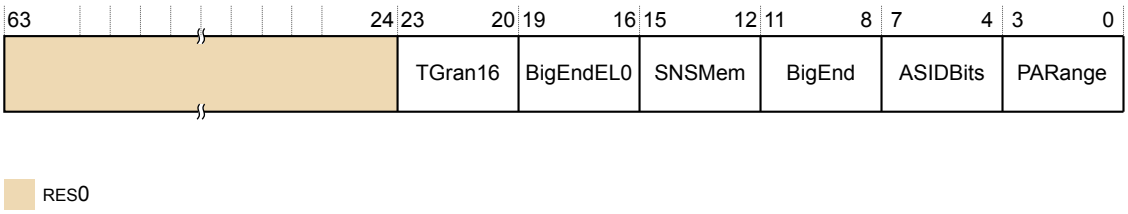


Figure B2-42 ID_AA64MMFR0_EL1 bit assignments

RES0, [63:24]

RES0 Reserved.

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:

0x2 40 bits, 1TB.The supported Physical Address Range is 40-bits. Other cores in the DSU may 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*.

B2.56 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.

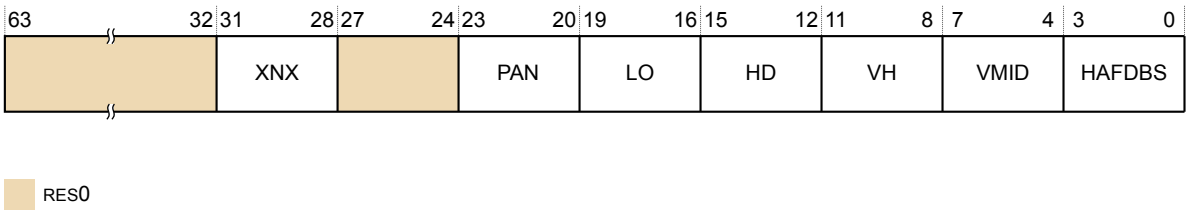


Figure B2-43 ID_AA64MMFR1_EL1 bit assignments

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*.

B2.57 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.

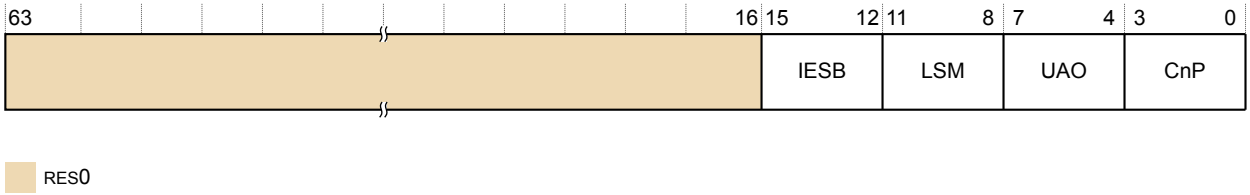


Figure B2-44 ID_AA64MMFR2_EL1 bit assignments

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*.

B2.58 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.

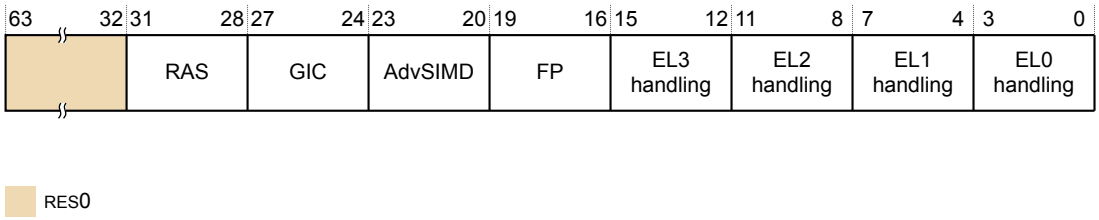


Figure B2-45 ID_AA64PFR0_EL1 bit assignments

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.

0xF Advanced SIMD is not implemented.

The FP and AdvSIMD both take the same value, as both must be implemented, or neither.

FP, [19:16]

Floating-point. The possible values are:

0x1 Floating-point, including Half-precision support, is implemented.

0xF Floating-point is not implemented.

The FP and AdvSIMD both take the same value, as both must be implemented, or neither.

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*.

B2.59 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-A55 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.

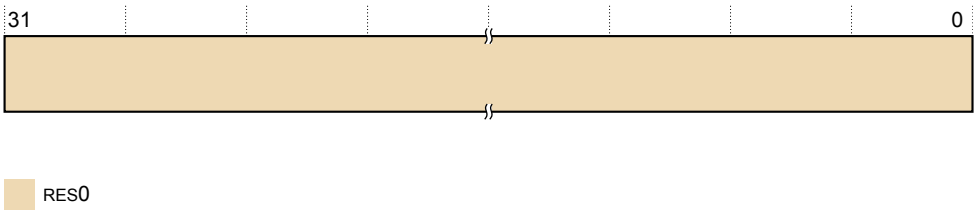


Figure B2-46 ID_AFR0_EL1 bit assignments

RES0, [31:0]

Reserved, RES0.

Configurations

AArch64 System register ID_AFR0_EL1 is architecturally mapped to AArch32 System register ID_AFR0. See *B1.56 ID_AFR0, Auxiliary Feature Register 0* 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*.

B2.60 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.

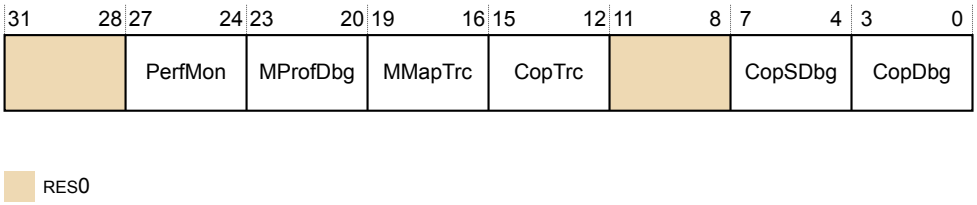


Figure B2-47 ID_DFR0_EL1 bit assignments

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 *B1.57 ID_DFR0, Debug Feature Register 0* on page B1-203.

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_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.

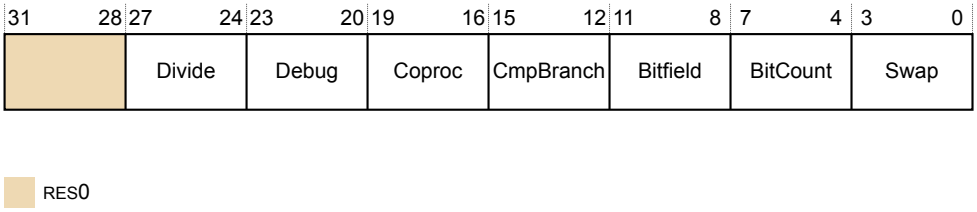


Figure B2-48 ID_ISAR0_EL1 bit assignments

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 *B1.58 ID_ISAR0, Instruction Set Attribute Register 0* on page B1-205.

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:

- *B2.62 ID_ISAR1_EL1, AArch32 Instruction Set Attribute Register 1, EL1* on page B2-370.
- *B2.63 ID_ISAR2_EL1, AArch32 Instruction Set Attribute Register 2, EL1* on page B2-372.
- *B2.64 ID_ISAR3_EL1, AArch32 Instruction Set Attribute Register 3, EL1* on page B2-374.
- *B2.65 ID_ISAR4_EL1, AArch32 Instruction Set Attribute Register 4, EL1* on page B2-376.
- *B2.66 ID_ISAR5_EL1, AArch32 Instruction Set Attribute Register 5, EL1* on page B2-378.

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_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.

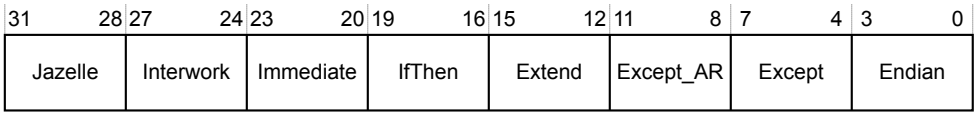


Figure B2-49 ID_ISAR1_EL1 bit assignments

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 *B1.59 ID_ISAR1, Instruction Set Attribute Register 1* on page B1-207.

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:

- *B2.61 ID_ISAR0_EL1, AArch32 Instruction Set Attribute Register 0, EL1* on page B2-368.
- *B2.63 ID_ISAR2_EL1, AArch32 Instruction Set Attribute Register 2, EL1* on page B2-372.
- *B2.64 ID_ISAR3_EL1, AArch32 Instruction Set Attribute Register 3, EL1* on page B2-374.
- *B2.65 ID_ISAR4_EL1, AArch32 Instruction Set Attribute Register 4, EL1* on page B2-376.
- *B2.66 ID_ISAR5_EL1, AArch32 Instruction Set Attribute Register 5, EL1* on page B2-378.

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_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.

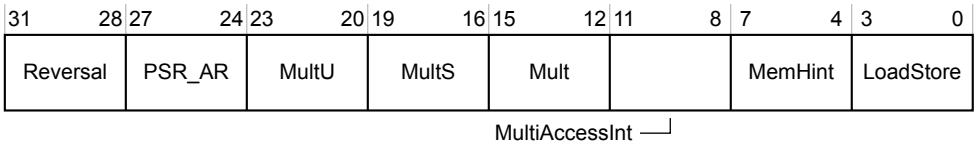


Figure B2-50 ID_ISAR2_EL1 bit assignments

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 `PLDW`instructions.

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 *B1.60 ID_ISAR2, Instruction Set Attribute Register 2* on page B1-209.

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:

- *B2.61 ID_ISAR0_EL1, AArch32 Instruction Set Attribute Register 0, EL1* on page B2-368.
- *B2.62 ID_ISAR1_EL1, AArch32 Instruction Set Attribute Register 1, EL1* on page B2-370.
- *B2.64 ID_ISAR3_EL1, AArch32 Instruction Set Attribute Register 3, EL1* on page B2-374.
- *B2.65 ID_ISAR4_EL1, AArch32 Instruction Set Attribute Register 4, EL1* on page B2-376.
- *B2.66 ID_ISAR5_EL1, AArch32 Instruction Set Attribute Register 5, EL1* on page B2-378.

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_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.

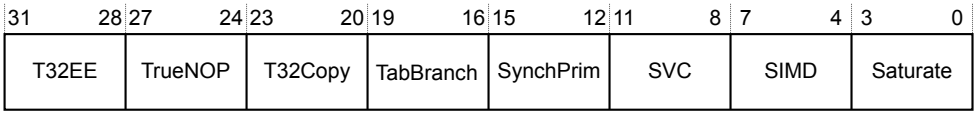


Figure B2-51 ID_ISAR3_EL1 bit assignments

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 *B1.61 ID_ISAR3, Instruction Set Attribute Register 3* on page B1-211.

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:
- *B2.61 ID_ISAR0_EL1, AArch32 Instruction Set Attribute Register 0, EL1* on page B2-368.
- *B2.62 ID_ISAR1_EL1, AArch32 Instruction Set Attribute Register 1, EL1* on page B2-370.
- *B2.63 ID_ISAR2_EL1, AArch32 Instruction Set Attribute Register 2, EL1* on page B2-372.
- *B2.65 ID_ISAR4_EL1, AArch32 Instruction Set Attribute Register 4, EL1* on page B2-376.
- *B2.66 ID_ISAR5_EL1, AArch32 Instruction Set Attribute Register 5, EL1* on page B2-378.

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_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.

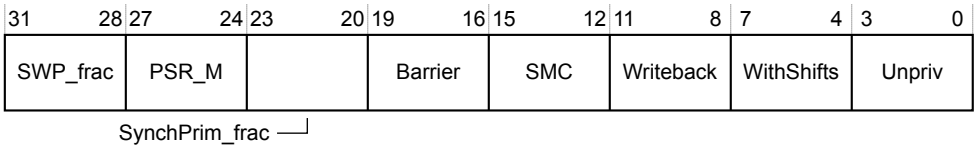


Figure B2-52 ID_ISAR4_EL1 bit assignments

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 *B1.62 ID_ISAR4, Instruction Set Attribute Register 4* on page B1-213.

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:

- *B2.61 ID_ISAR0_EL1, AArch32 Instruction Set Attribute Register 0, EL1* on page B2-368.
- *B2.62 ID_ISAR1_EL1, AArch32 Instruction Set Attribute Register 1, EL1* on page B2-370.
- *B2.63 ID_ISAR2_EL1, AArch32 Instruction Set Attribute Register 2, EL1* on page B2-372.
- *B2.64 ID_ISAR3_EL1, AArch32 Instruction Set Attribute Register 3, EL1* on page B2-374.
- *B2.66 ID_ISAR5_EL1, AArch32 Instruction Set Attribute Register 5, EL1* on page B2-378.

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_ISAR5_EL1, AArch32 Instruction Set Attribute Register 5, EL1

The ID_ISAR5_EL1 provides information about the instruction sets that the core implements.

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_ISAR5_EL1 is a 32-bit register, and is part of the Identification registers functional group.

This register is Read Only.

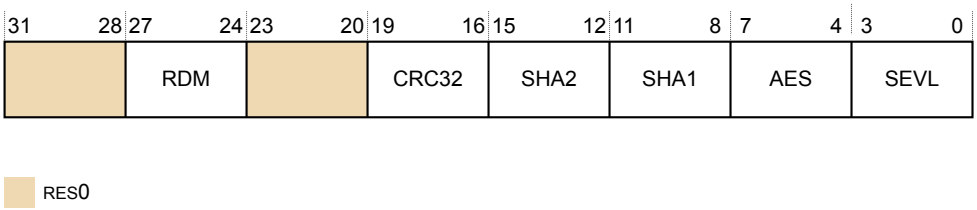


Figure B2-53 ID_ISAR5_EL1 bit assignments

[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 *B1.63 ID_ISAR5, Instruction Set Attribute Register 5* on page B1-215.

ID_ISAR5 must be interpreted with ID_ISAR0_EL1, ID_ISAR1_EL1, ID_ISAR2_EL1, ID_ISAR3_EL1, and ID_ISAR4_EL1. See:

- *B2.61 ID_ISAR0_EL1, AArch32 Instruction Set Attribute Register 0, EL1* on page B2-368.
- *B2.62 ID_ISAR1_EL1, AArch32 Instruction Set Attribute Register 1, EL1* on page B2-370.
- *B2.63 ID_ISAR2_EL1, AArch32 Instruction Set Attribute Register 2, EL1* on page B2-372.
- *B2.64 ID_ISAR3_EL1, AArch32 Instruction Set Attribute Register 3, EL1* on page B2-374.
- *B2.65 ID_ISAR4_EL1, AArch32 Instruction Set Attribute Register 4, EL1* on page B2-376.

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_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.

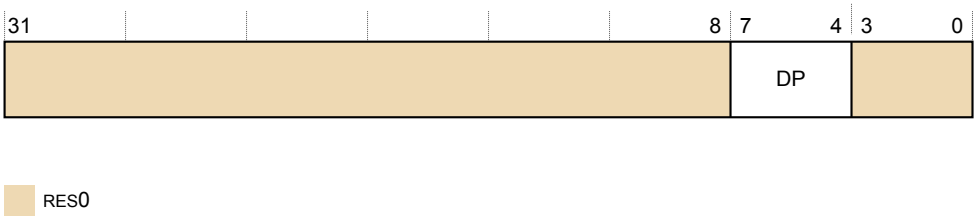


Figure B2-54 ID_ISAR6_EL1 bit assignments

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 *B1.64 ID_ISAR6, Instruction Set Attribute Register 6* on page B1-217.

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:

- *B2.61 ID_ISAR0_EL1, AArch32 Instruction Set Attribute Register 0, EL1* on page B2-368.
- *B2.62 ID_ISAR1_EL1, AArch32 Instruction Set Attribute Register 1, EL1* on page B2-370.
- *B2.63 ID_ISAR2_EL1, AArch32 Instruction Set Attribute Register 2, EL1* on page B2-372.
- *B2.64 ID_ISAR3_EL1, AArch32 Instruction Set Attribute Register 3, EL1* on page B2-374.
- *B2.65 ID_ISAR4_EL1, AArch32 Instruction Set Attribute Register 4, EL1* on page B2-376.
- *B2.66 ID_ISAR5_EL1, AArch32 Instruction Set Attribute Register 5, EL1* on page B2-378.

Bit fields and details not provided in this description are architecturally defined. See the *ARM® Architecture Reference Manual ARMv8, for ARMv8-A architecture profile*.

B2.68 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.

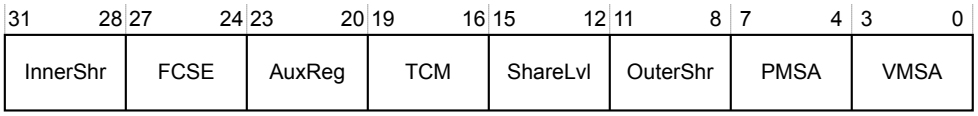


Figure B2-55 ID_MMFR0_EL1 bit assignments

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 *B1.65 ID_MMFR0, Memory Model Feature Register 0* on page B1-218.

Must be interpreted with ID_MMFR1_EL1, ID_MMFR2_EL1, ID_MMFR3_EL1, and ID_MMFR4_EL1. See:

- *B2.69 ID_MMFR1_EL1, AArch32 Memory Model Feature Register 1, EL1* on page B2-383.
- *B2.70 ID_MMFR2_EL1, AArch32 Memory Model Feature Register 2, EL1* on page B2-385.
- *B2.71 ID_MMFR3_EL1, AArch32 Memory Model Feature Register 3, EL1* on page B2-387.
- *B2.72 ID_MMFR4_EL1, AArch32 Memory Model Feature Register 4, EL1* on page B2-389.

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_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.

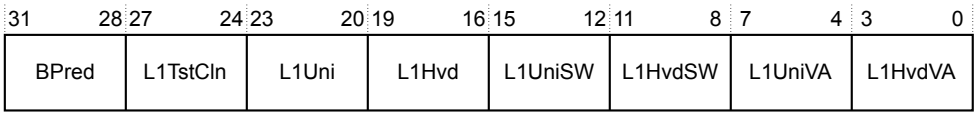


Figure B2-56 ID_MMFR1_EL1 bit assignments

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 *B1.66 ID_MMFR1, Memory Model Feature Register 1* on page B1-220.

Must be interpreted with ID_MMFR0_EL1, ID_MMFR2_EL1, ID_MMFR3_EL1, and ID_MMFR4_EL1. See:

- *B2.68 ID_MMFR0_EL1, AArch32 Memory Model Feature Register 0, EL1* on page B2-381.
- *B2.70 ID_MMFR2_EL1, AArch32 Memory Model Feature Register 2, EL1* on page B2-385.
- *B2.71 ID_MMFR3_EL1, AArch32 Memory Model Feature Register 3, EL1* on page B2-387.
- *B2.72 ID_MMFR4_EL1, AArch32 Memory Model Feature Register 4, EL1* on page B2-389.

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_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.

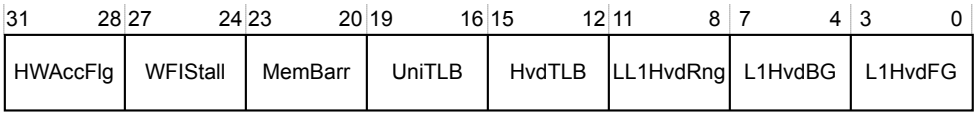


Figure B2-57 ID_MMFR2_EL1 bit assignments

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 *B1.67 ID_MMFR2, Memory Model Feature Register 2* on page B1-222.

Must be interpreted with ID_MMFR0_EL1, ID_MMFR1_EL1, ID_MMFR3_EL1, and ID_MMFR4_EL1. See:

- *B2.68 ID_MMFR0_EL1, AArch32 Memory Model Feature Register 0, EL1* on page B2-381.
- *B2.69 ID_MMFR1_EL1, AArch32 Memory Model Feature Register 1, EL1* on page B2-383.
- *B2.71 ID_MMFR3_EL1, AArch32 Memory Model Feature Register 3, EL1* on page B2-387.
- *B2.72 ID_MMFR4_EL1, AArch32 Memory Model Feature Register 4, EL1* on page B2-389.

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*.

B2.71 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.

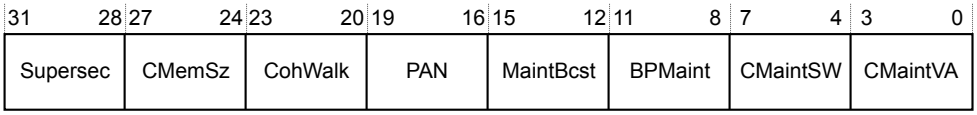


Figure B2-58 ID_MMFR3_EL1 bit assignments

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 *B1.68 ID_MMFR3, Memory Model Feature Register 3* on page B1-224.

Must be interpreted with ID_MMFR0_EL1, ID_MMFR1_EL1, ID_MMFR2_EL1, and ID_MMFR4_EL1. See:

- *B2.68 ID_MMFR0_EL1, AArch32 Memory Model Feature Register 0, EL1* on page B2-381.
- *B2.69 ID_MMFR1_EL1, AArch32 Memory Model Feature Register 1, EL1* on page B2-383.
- *B2.70 ID_MMFR2_EL1, AArch32 Memory Model Feature Register 2, EL1* on page B2-385.
- *B2.72 ID_MMFR4_EL1, AArch32 Memory Model Feature Register 4, EL1* on page B2-389.

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_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.

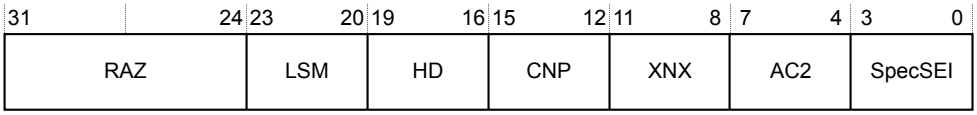


Figure B2-59 ID_MMFR4_EL1 bit assignments

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 *B1.69 ID_MMFR4, Memory Model Feature Register 4* on page B1-226.

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:

- *B2.68 ID_MMFR0_EL1, AArch32 Memory Model Feature Register 0, EL1* on page B2-381.
- *B2.69 ID_MMFR1_EL1, AArch32 Memory Model Feature Register 1, EL1* on page B2-383.
- *B2.70 ID_MMFR2_EL1, AArch32 Memory Model Feature Register 2, EL1* on page B2-385.
- *B2.71 ID_MMFR3_EL1, AArch32 Memory Model Feature Register 3, EL1* on page B2-387.

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_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.

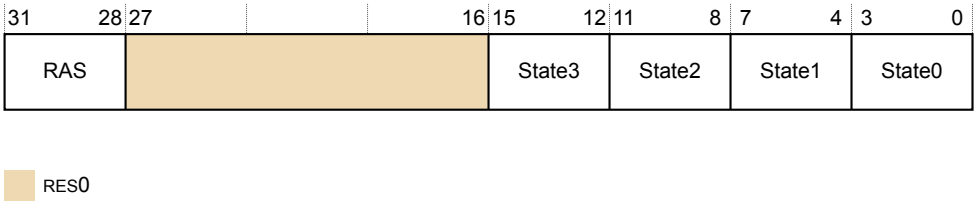


Figure B2-60 ID_PFR0_EL1 bit assignments

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 *B1.70 ID_PFR0, Processor Feature Register 0* on page B1-228.

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_PFR1_EL1, AArch32 Processor 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 must be interpreted with ID_PFR0. It is part of the Identification registers functional group.

This register is Read Only.

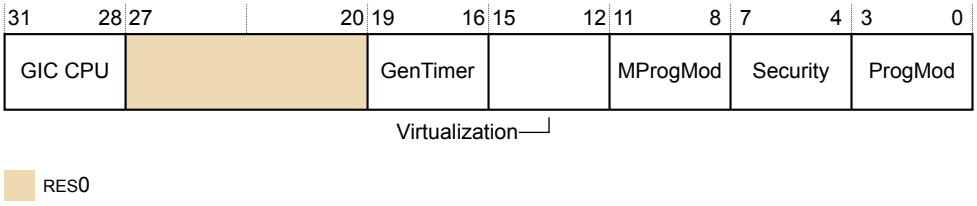


Figure B2-61 ID_PFR1_EL1 bit assignments

GIC CPU, [31:28]
GIC CPU support:

`0x0` GIC CPU interface is disabled, **GICCDISABLE** is HIGH.

`0x1` GIC CPU interface is implemented and 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_EL1 is architecturally mapped to AArch32 register ID_PFR1. See *B1.71 ID_PFR1, Processor Feature Register 1* on page B1-229.

Bit fields and details not provided in this description are architecturally defined. See the *ARM® Architecture Reference Manual ARMv8, for ARMv8-A architecture profile*.

B2.75 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.

- *B2.75.1 IFSR32_EL2 with Short-descriptor translation table format* on page B2-394.
- *B2.75.2 IFSR32_EL2 with Long-descriptor translation table format* on page B2-394.

Configurations

IFSR32_EL2 is architecturally mapped to AArch32 register IFSR(NS). See *B1.72 IFSR, Instruction Fault Status Register* on page B1-231.

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:

- *B2.75.1 IFSR32_EL2 with Short-descriptor translation table format* on page B2-394.
- *B2.75.2 IFSR32_EL2 with Long-descriptor translation table format* on page B2-394.

B2.75.1 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:

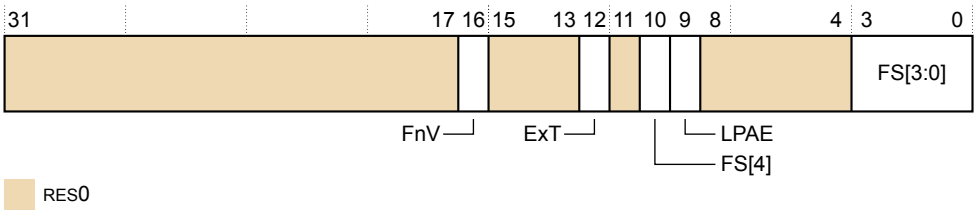


Figure B2-62 IFSR32_EL2 bit assignments for Short-descriptor translation table format

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*.

B2.75.2 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:

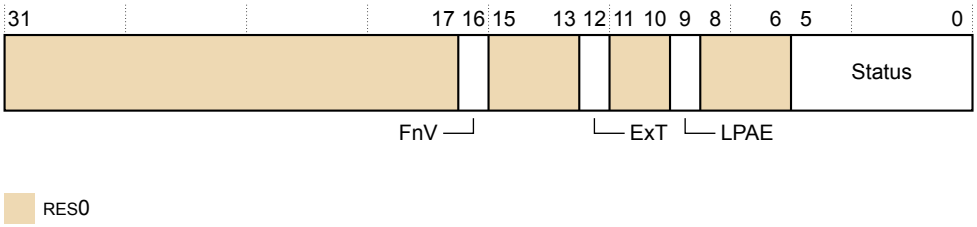


Figure B2-63 IFSR32_EL2 bit assignments for Long-descriptor translation table format

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*.

B2.76 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.

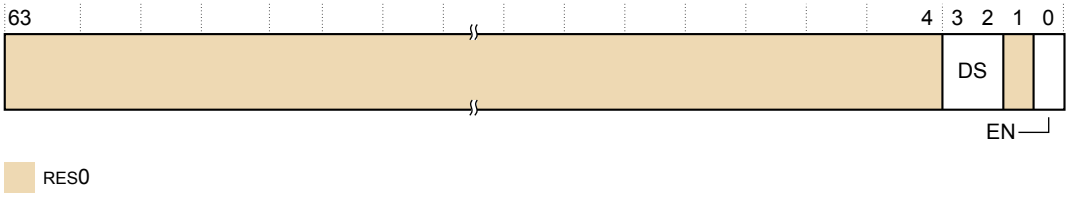


Figure B2-64 LORC_EL1 bit assignments

[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*.

B2.77 LOREA_EL1, LORegion End Address Register, EL1

The LOREA_EL1 register holds the physical address of the end of the LORegion described in the current LORegion descriptor selected by LORC_EL1.DS.

Bit field descriptions

LOREA_EL1 is a 64-bit register and is part of the Virtual memory control registers functional group.

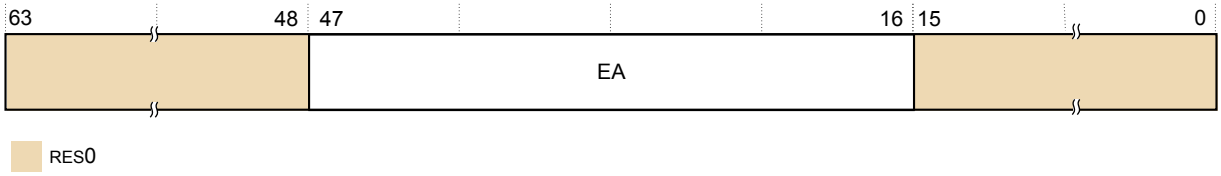


Figure B2-65 LOREA_EL1 bit assignments

[63:48]
Reserved, RES0.

EA, [47:16]
End physical address bits.

[15:0]
Reserved, RES0.

Configurations
The LOREA_EL1 register is only applicable to the AArch64 state and is not accessible at any exception level in AArch32.

If no LORegion descriptors are supported by the core, then this register is RES0.

If LORC_EL1.DS points to a LORegion that is not supported by the core, then this register is RES0.

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*.

B2.78 LORID_EL1, Limited Order Region Identification Register, EL1

The LORID_EL1 indicates the number of LORegions and LORegion descriptors supported by the Cortex-A55 core.

Bit field descriptions

LORID_EL1 is a 64-bit register, and is part of the Virtual memory control registers functional group.

This register is Read Only.

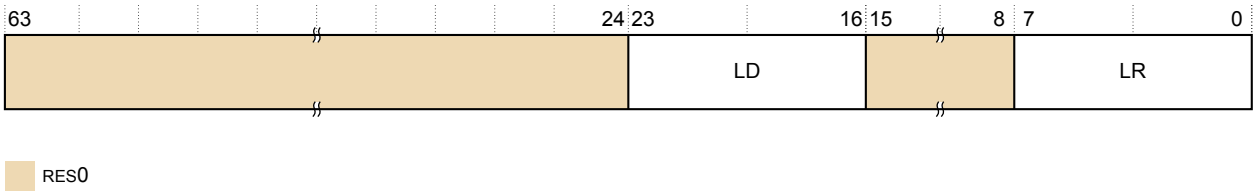


Figure B2-66 LORID_EL1 bit assignments

RES0, [63:24]

Reserved, RES0.

LD, [23:16]

Indicates the number of LOR Descriptors supported by the core. The value is:

`0x4` Four LOR Descriptors supported.

RES0, [15:8]

Reserved, RES0.

LR, [7:0]

Indicates the number of LORegions supported by the core. The value is:

`0x4` Four LORegions supported.

Configurations

The LORID_EL1 is only applicable to the AArch64 state and is not accessible at any exception level in AArch32.

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 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.

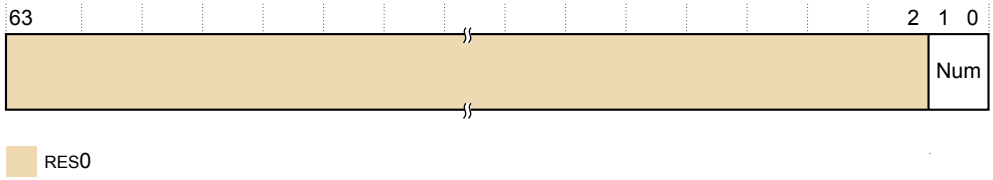


Figure B2-67 LORN_EL1 bit assignments

[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*.

B2.80 LORSA_EL1, LORegion Start Address Register, EL1

The LORSA_EL1 register indicates whether the current LORegion descriptor selected by LORC_EL1.DS is is enabled, and holds the physical address of the start of the LORegion.

Bit field descriptions

LORSA_EL1 is a 64-bit register and is part of the Virtual memory control registers functional group.

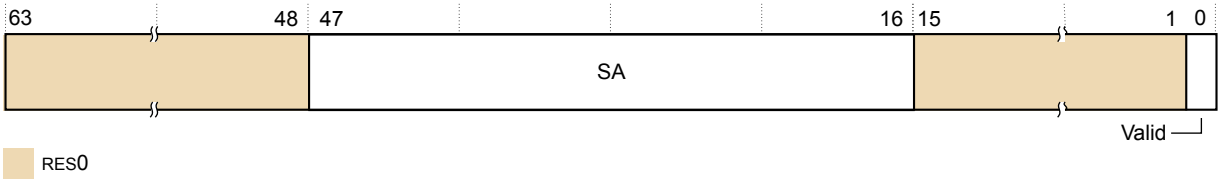


Figure B2-68 LORSA_EL1 bit assignments

RES0, [63:48]
Reserved, RES0.

SA, [47:16]
Start physical address bits.

RES0, [15:1]
Reserved, RES0.

Valid, [0]
Valid. Indicates whether the LORegion Descriptor is enabled. The possible values are:

`0` Not valid. This is the reset value.

`1` Valid.

Configurations

The LORSA_EL1 register is only applicable to the AArch64 state and is not accessible at any exception level in AArch32.

If no LORegion descriptors are supported by the core, then this register is RES0.

If LORC_EL1.DS points to a LORegion that is not supported by the core, then this register is RES0.

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*.

B2.81 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.

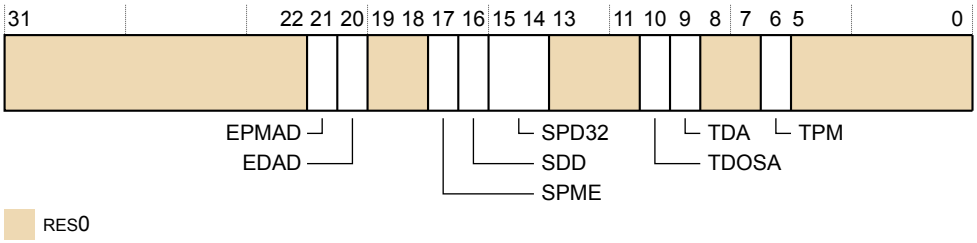


Figure B2-69 MDCR_EL3 bit assignments

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*.

B2.82 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.

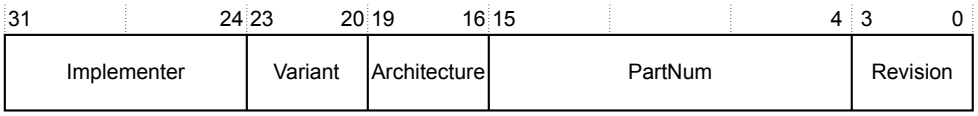


Figure B2-70 MIDR_EL1 bit assignments

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:

`0x1` r1p0.

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:

`0xD05` Cortex-A55 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:

`0x0` r1p0.

Configurations
The MIDR_EL1 is:

- Architecturally mapped to the AArch32 MIDR register. See *B1.73 MIDR, Main ID Register* on page B1-233.
- 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*.

B2.83 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.

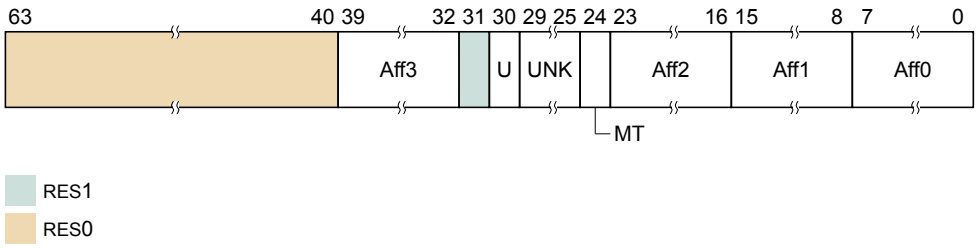


Figure B2-71 MPIDR_EL1 bit assignments

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-A55 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-A55 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-A55 core is single-threaded, so this field has the value `0x00`.

Configurations

MPIDR_EL1[31:0] is:

- Architecturally mapped to AArch32 register MPIDR. See *B1.74 MPIDR, Multiprocessor Affinity Register* on page B1-234.
- 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*.

B2.84 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.

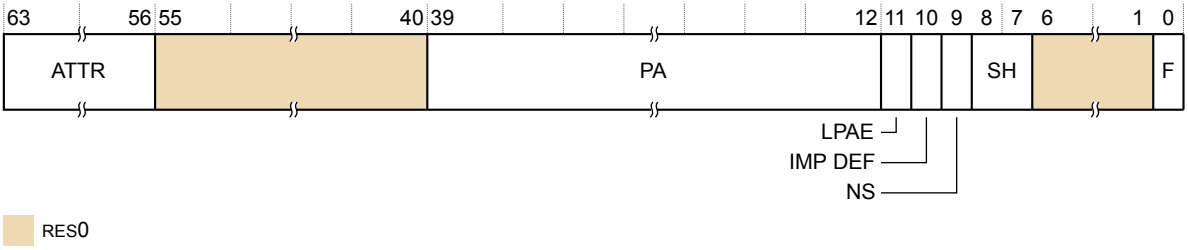


Figure B2-72 PAR bit assignments, PAR_EL1.F is 0

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*.

B2.85 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-A55 core.

Bit field descriptions

REVIDR_EL1 is a 32-bit register, and is part of the Identification registers functional group.

This register is Read Only.

Figure B2-73 REVIDR_EL1 bit assignments

IMPLEMENTATION DEFINED, [31:0]

IMPLEMENTATION DEFINED.

Configurations

REVIDR_EL1 is architecturally mapped to AArch32 register REVIDR. See *B1.76 REVIDR, Revision ID Register* on page B1-241.

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*.

B2.86 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.

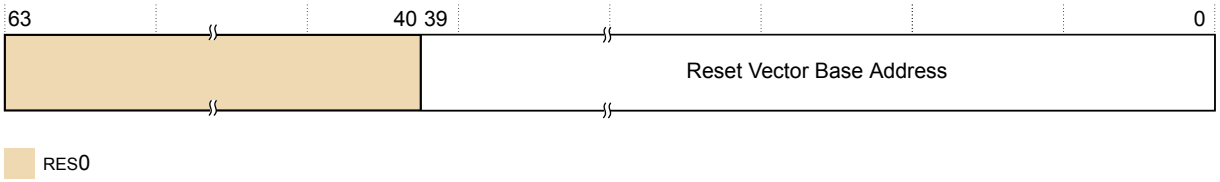


Figure B2-74 RVBAR_EL3 bit assignments

RES0, [63:40]

Reserved, RES0.

RVBA, [39:0]

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.

The Reset Vector Base Address is determined by the signal **RVBARADDRx**.

Configurations

There is no configuration information.

Bit fields and details not provided in this description are architecturally defined. See the *ARM® Architecture Reference Manual ARMv8, for ARMv8-A architecture profile*.

B2.87 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.

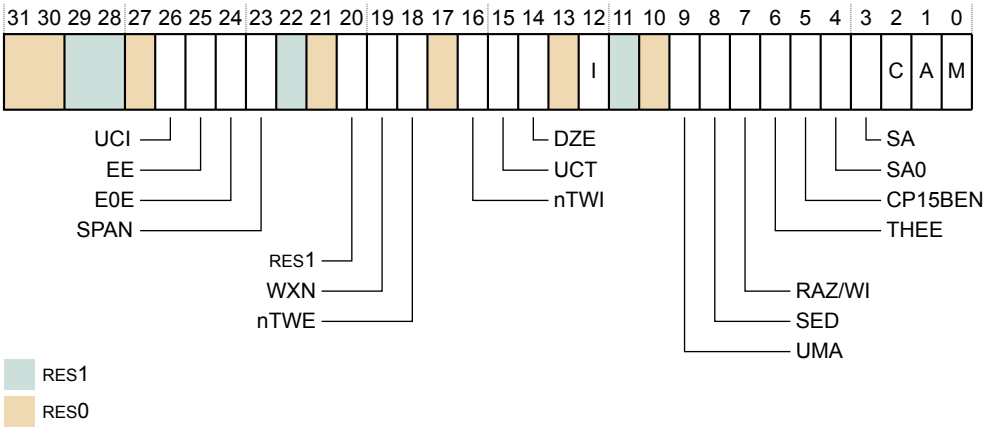


Figure B2-75 SCTLR_EL1 bit assignments

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 signal.

E0E, [24]

Endianness of explicit data access at EL0. The possible values are:

`0` Explicit data accesses at EL0 are little-endian. This is reset value.

`1` Explicit data accesses at EL0 are big-endian.

SED, [8]

SETEND instruction disable. The possible values are:

`0` The SETEND instruction is enabled. This is the reset value.

`1` The SETEND instruction is UNDEFINED.

Configurations

SCTLR_EL1 is architecturally mapped to AArch32 register SCTLR(NS) See *B1.78 SCTLR, System Control Register* on page B1-243.

Bit fields and details not provided in this description are architecturally defined. See the *ARM® Architecture Reference Manual ARMv8, for ARMv8-A architecture profile*.

B2.88 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.

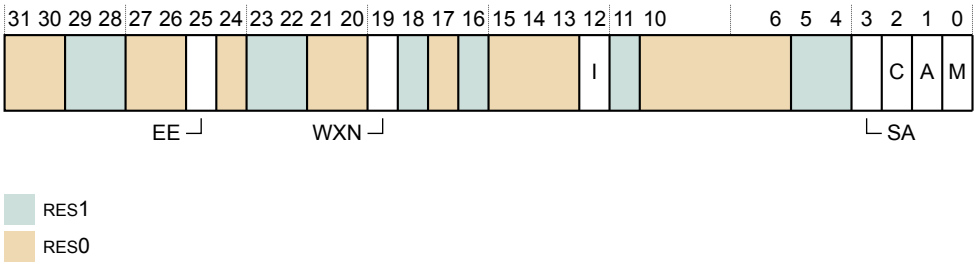


Figure B2-76 SCTLR_EL2 bit assignments

Apart from bits [12], [2], and [0], this register resets to UNKNOWN values.

Configurations

SCTLR_EL2 is architecturally mapped to AArch32 register HSCTLR. See *B1.53 HSCTLR, Hyp System Control Register* on page B1-197.

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*.

B2.89 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.

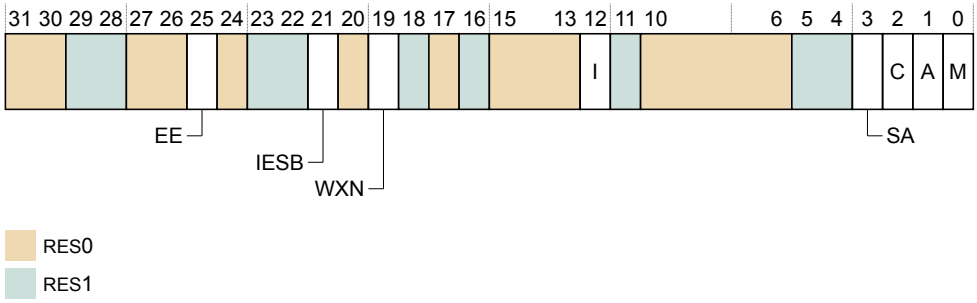


Figure B2-77 SCTLR_EL3 bit assignments

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. This is the reset value.

`1` Big endian.

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 not provided in this description are architecturally defined. See the *ARM® Architecture Reference Manual ARMv8, for ARMv8-A architecture profile*.

B2.90 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.

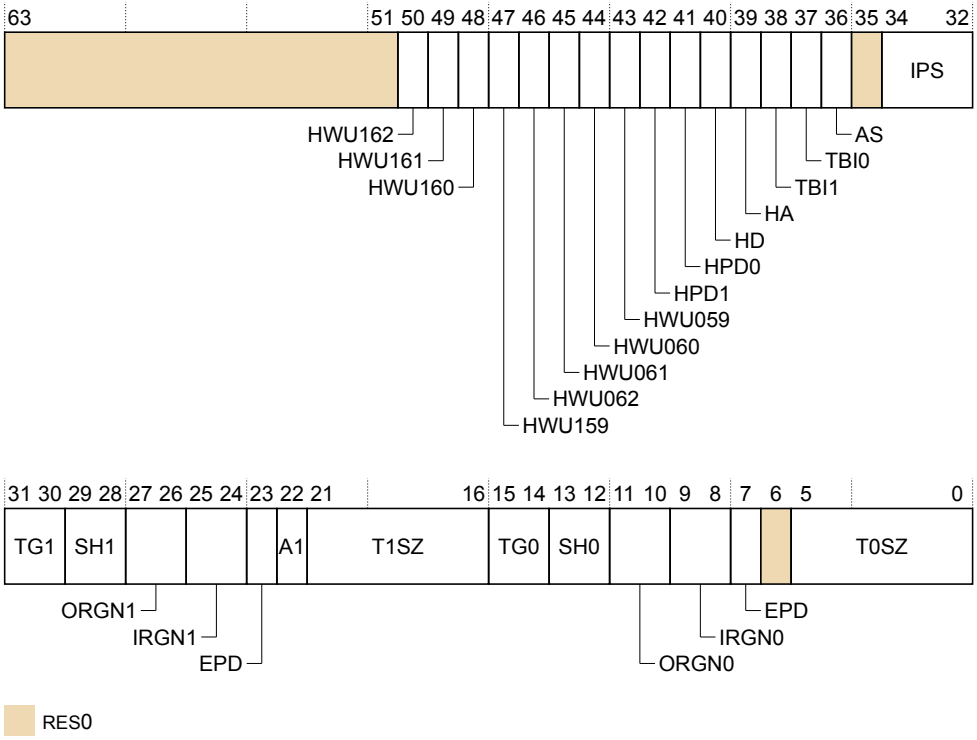


Figure B2-78 TCR_EL1 bit assignments

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*.

B2.91 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 64-bit register, and is part of:

- The Virtualization registers functional group.
- The Virtual memory control registers functional group.

The register has configurations of bit assignment dependent upon whether the core is running a hypervisor or running a host Host Operating System. This is controlled by the value of the E2H bit in register HCR_EL2:

`0` Hpervisor configuration, see *When HCR_EL2.E2H==0* on page B2-413.

`1` Host Operating System configuration: HCR_EL2.E2H==1 Address translation aborted, see *When HCR_EL2.E2H==1* on page B2-415.

Configurations

TCR_EL2 is architecturally mapped to AArch32 register HTCR. See the *ARM® Architecture Reference Manual ARMv8, for ARMv8-A architecture profile*.

If EL2 is not implemented, this register is RES0 from EL3.

RW fields in this register reset to architecturally UNKNOWN values.

When HCR_EL2.E2H==0

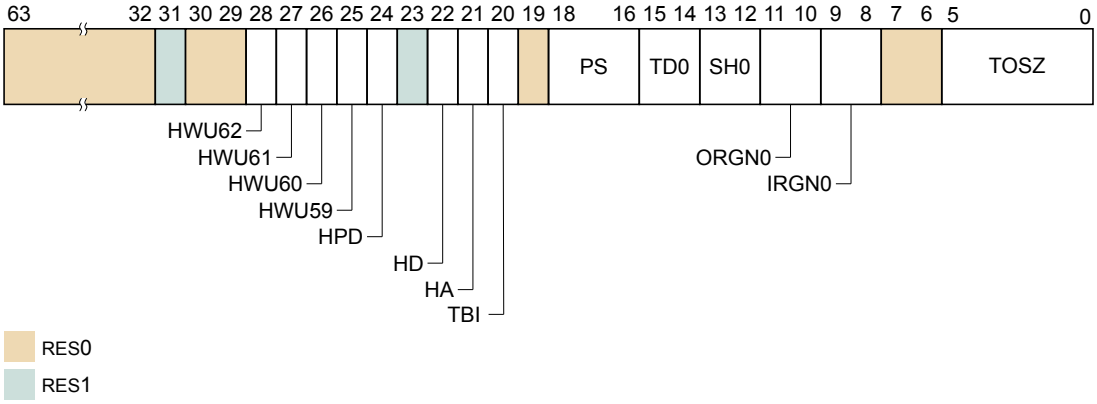


Figure B2-79 TCR_EL2 bit assignments when HCR_EL2.E2H==0

HPD, [24]

Hierarchical Permission Disables. This affects the hierarchical control bits, APTable, PXNTable, and UXNTable, except NSTable, in the translation tables pointed to by TTBR0_EL2. The possible values are:

`0` Hierarchical Permissions are enabled.

`1` Hierarchical Permissions are disabled.

Note

In this case bit[61] (APTable[0]) and bit[59] (PXNTable) of the next level descriptor attributes are required to be to be ignored by the PE, and are no longer reserved, allowing them to be used by software.

HD, [22]

Hardware management of dirty state in stage 1 translations from EL2. 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.

Implementation of this bit is OPTIONAL, and, if not implemented, this bit is RES0.

HA, [21]

Hardware Access flag update in stage 1 translations from EL2. The possible values are:

`0` Stage 1 Access flag update disabled.

`1` Stage 1 Access flag update enabled.

PS, [18:16]

Physical address size. The possible values are:

`0b000` 32 bits, 4GB.

`0b001` 36 bits, 64GB.

`0b010` 40 bits, 1TB.

Other values are reserved.

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.

SH0, [13:12]

Shareability attribute for memory associated with translation table walks using TTBR0_EL2.

The possible values are:

`0b00` Non-shareable.

`0b01` Reserved.

`0b10` Outer shareable.

`0b11` Inner shareable.

When HCR_EL2.E2H==1

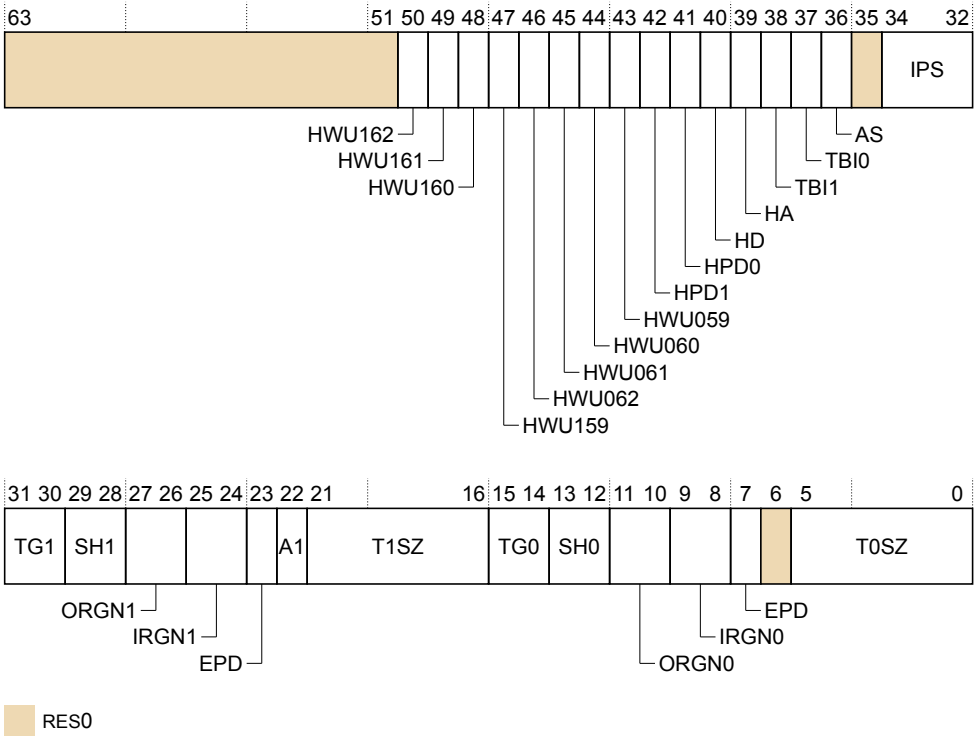


Figure B2-80 TCR_EL2 bit assignments when HCR_EL2.E2H==1

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.

Implementation of this bit is OPTIONAL, and, if not implemented, this bit is RES0.

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.

Implementation of this bit is OPTIONAL, and, if not implemented, this bit is RES0.

TG0, [15:14]

TTBR0_EL1 granule size. The possible values are:

`0b00` 4KB.

`0b10` 16KB.

`0b01` 64KB.

`0b11` Reserved.

All other values are not supported.

SH0, [13:12]

Shareability attribute for memory associated with translation table walks using TTBR0_EL1. The possible values are:

`0b00` Non-shareable.

`0b01` Reserved.

`0b10` Outer shareable.

`0b11` Inner shareable.

Bit fields and details not provided in this description are architecturally defined. See the *ARM® Architecture Reference Manual ARMv8, for ARMv8-A architecture profile*.

B2.92 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.

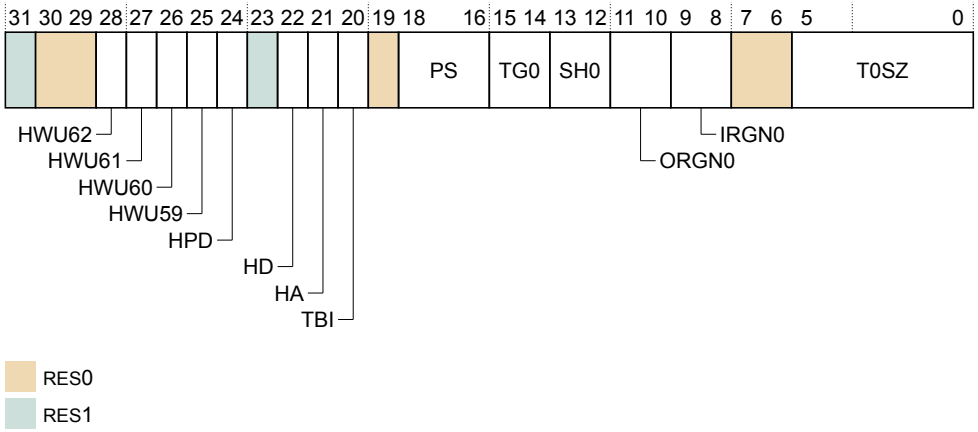


Figure B2-81 TCR_EL3 bit assignments

HPD, [24]

Hierarchical Permission Disables. This affects the hierarchical control bits, APTable, PXNTable, and UXNTable, except NSTable, in the translation tables pointed to by TTBR0_EL3. The possible values are:

`0` Hierarchical Permissions are enabled.

`1` Hierarchical Permissions are disabled.

Note

In this case bit[61] (APTable[0]) and bit[59] (PXNTable) of the next level descriptor attributes are required to be to be ignored by the PE, and are no longer reserved, allowing them to be used by software.

HD, [22]

Hardware management of dirty state in stage 1 translations from EL3. 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.

Implementation of this bit is OPTIONAL, and, if not implemented, this bit is RES0.

HA, [21]

Hardware Access flag update in stage 1 translations from EL3. The possible values are:

`0` Stage 1 Access flag update disabled.

`1` Stage 1 Access flag update enabled.

PS, [18:16]

Physical address size. The possible values are:

`0b000` 32 bits, 4GB.

`0b001` 36 bits, 64GB.

`0b010` 40 bits, 1TB.

Other values are reserved.

TG0, [15:14]

TTBR0_EL3 granule size. The possible values are:

`0b00` 4KB.

`0b10` 16KB.

`0b01` 64KB.

`0b11` Reserved.

All other values are not supported.

SH0, [13:12]

Shareability attribute for memory associated with translation table walks using TTBR0_EL3.

The possible values are:

`0b00` Non-shareable.

`0b01` Reserved.

`0b10` Outer shareable.

`0b11` Inner shareable.

Bit fields and details not provided in this description are architecturally defined. See the *ARM® Architecture Reference Manual ARMv8, for ARMv8-A architecture profile*.

B2.93 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.

This register is used when HCR_EL2.E2H is 0.

Note

When HCR_EL2.E2H is 1, TTBR0_EL2 is used.

Bit field descriptions

TTBR0_EL1 is a 64-bit register, and is part of the Virtual memory control registers functional group.

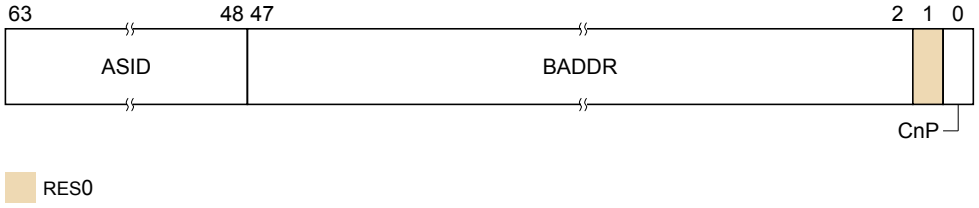


Figure B2-82 TTBR0_EL1 bit assignments

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, [47:2]
Translation table base address.

RES0, [1]

RES0 Reserved.

CnP, [0]
Common not Private. Supports selective sharing of TLB entries across multiple cores. The value is:

`0` CnP is not supported.

`1` CnP is supported.

Configurations
TTBR0_EL1 is architecturally mapped to AArch32 register TTBR0. See *B1.82 TTBR0, Translation Table Base Register 0* on page B1-255.RW fields in this register reset to architecturally UNKNOWN values.

Any of the fields in this register are permitted to be cached in a TLB.

Bit fields and details not provided in this description are architecturally defined. See the *ARM® Architecture Reference Manual ARMv8, for ARMv8-A architecture profile*.

B2.94 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.

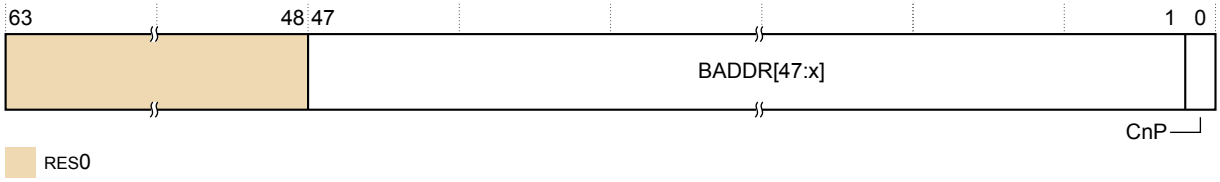


Figure B2-83 TTBR0_EL2 bit assignments

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*.

B2.95 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, and is part of the Virtual memory control registers functional group.

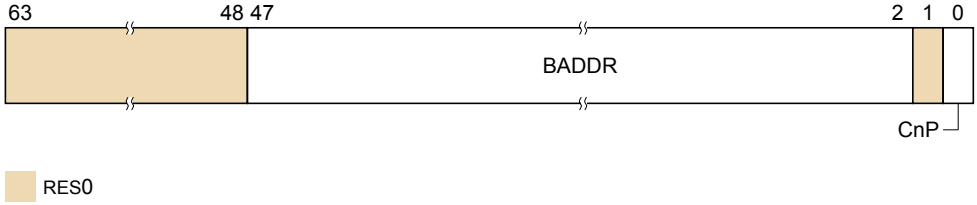


Figure B2-84 TTBR0_EL3 bit assignments

RES0, [63:48]

RES0 Reserved.

BADDR, [47:2]

Translation table base address.

RES0, [1]

RES0 Reserved.

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 *B1.82 TTBR0, Translation Table Base Register 0* on page B1-255.

RW fields in this register reset to architecturally UNKNOWN values.

Any of the fields in this register are permitted to be cached in a TLB.

Bit fields and details not provided in this description are architecturally defined. See the *ARM® Architecture Reference Manual ARMv8, for ARMv8-A architecture profile*.

B2.96 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.

This register is used when HCR_EL2.E2H is 0.

Note

When HCR_EL2.E2H is 1, TTBR1_EL2 is used.

Bit field descriptions

TTBR1_EL1 is a 64-bit register, and is part of the Virtual memory control registers functional group.

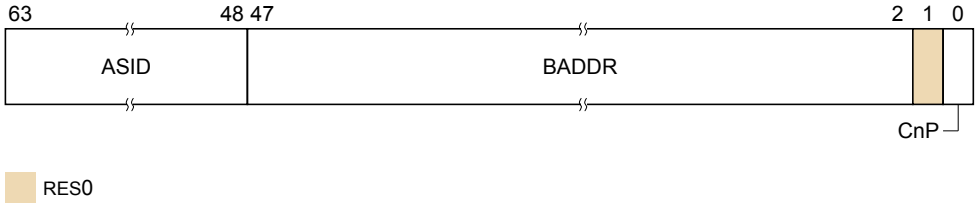


Figure B2-85 TTBR1_EL1 bit assignments

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, [47:2]
Translation table base address.

RES0, [1]

RES0 Reserved.

CnP, [0]
Common not Private. Supports selective sharing of TLB entries across multiple cores. 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 *B1.83 TTBR1, Translation Table Base Register 1* on page B1-257.RW fields in this register reset to architecturally UNKNOWN values.

Any of the fields in this register are permitted to be cached in a TLB.

Bit fields and details not provided in this description are architecturally defined. See the *ARM® Architecture Reference Manual ARMv8, for ARMv8-A architecture profile*.

B2.97 TTBR1_EL2, Translation Table Base Register 1, EL2

TTBR1_EL2 has the same format and contents as TTBR1_EL1.

See *B2.96 TTBR1_EL1, Translation Table Base Register 1, EL1* on page B2-422.

B2.98 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 *B2.98.2 VDISR_EL2 with short-descriptor translation table format* on page B2-425.
- When written at EL1 using long-descriptor format. See *B2.98.1 VDISR_EL2 with long-descriptor translation table format* on page B2-424.
- When written at EL2. See *B2.98.3 VDISR_EL2 at EL1 using AArch64* on page B2-426.

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 *B1.85 VDISR, Virtual Deferred Interrupt Status Register* on page B1-260.

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:

B2.98.1 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.

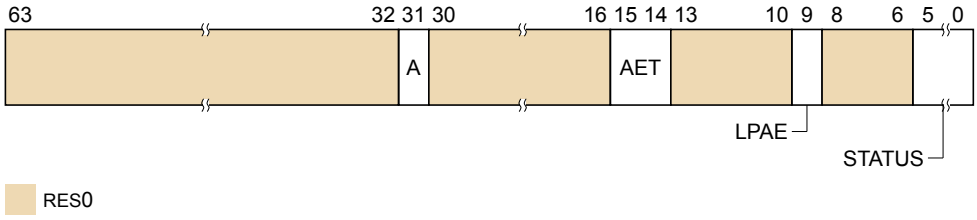


Figure B2-86 VDISR_EL2 bit assignments for the Long-descriptor format

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, [13:10]

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.

B2.98.2 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.

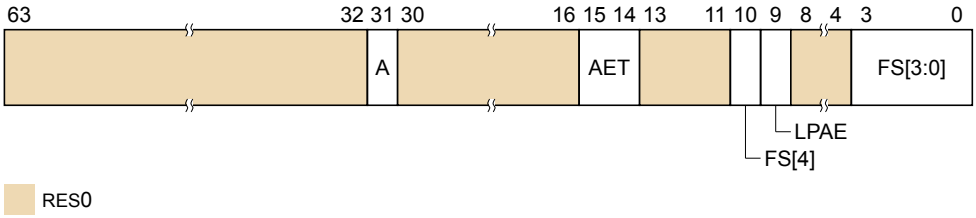


Figure B2-87 VDISR_EL2 bit assignments for Short-descriptor translation table format

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:

`0b01` *Uncorrected error, Unrecoverable error* (UEU).

RES0, [13:11]

RES0 Reserved.

FS[4], [10]
Bit 4 of 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.

FS[3:0], [3:0]
Bits [3:0] of Fault status code. Set to `0b10110` when ESB defers a virtual SError interrupt. The value of this field is:

`0b10110` Asynchronous SError interrupt.

B2.98.3 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:

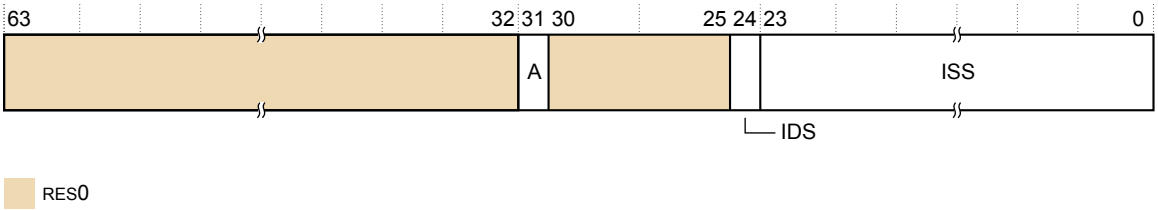


Figure B2-88 VDISR_EL2 at EL1 using AArch64

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].

B2.99 VMPIDR_EL2, Virtualization Multiprocessor ID Register, EL2

The VMPIDR_EL2 provides the value of the Virtualization Multiprocessor ID. This is the value returned by Non-secure EL1 reads of MPIDR.

Bit field descriptions

VMPIDR_EL2 is a 64-bit register, and is part of:

- The Identification registers functional group.
- The Virtualization registers functional group.

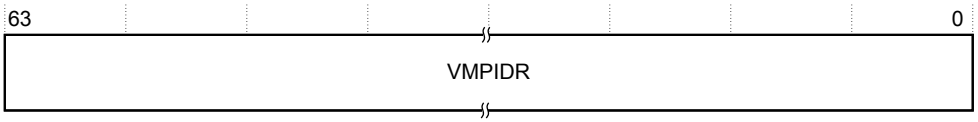


Figure B2-89 VMPIDR_EL2 bit assignments

VMPIDR, [63:0]

MPIDR value returned by Non-secure EL1 reads of the MPIDR_EL1. The MPIDR description defines the subdivision of this value. See *Figure B2-71 MPIDR_EL1 bit assignments* on page B2-404.

Configurations

VMPIDR_EL2[31:0] is architecturally mapped to AArch32 register VMPIDR. See *B1.86 VMPIDR, Virtualization Multiprocessor ID Register* on page B1-263.

If EL2 is not implemented, reads of this register return the value of the MPIDR_EL1, and writes to the register are ignored.

RW fields in this register reset to architecturally UNKNOWN values.

VMPIDR_EL2 resets to the value of MPIDR_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*.

B2.100 VPIDR_EL2, Virtualization Processor ID Register, EL2

The VPIDR_EL2 holds the value of the Virtualization Processor ID. This is the value returned by Non-secure EL1 reads of MIDR_EL1.

Bit field descriptions

VPIDR_EL2 is a 32-bit register, and is part of:

- The Identification registers functional group.
- The Virtualization registers functional group.

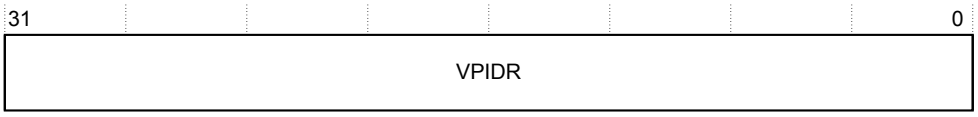


Figure B2-90 VPIDR_EL2 bit assignments

VPIDR, [31:0]

MIDR_EL1 value returned by Non-secure EL1 reads of the MIDR_EL1. The MIDR_EL1 description defines the subdivision of this value. See *Figure B2-70 MIDR_EL1 bit assignments* on page B2-403.

Configurations

VPIDR_EL2 is architecturally mapped to AArch32 register VPIDR. See *B1.87 VPIDR, Virtualization Processor ID Register* on page B1-264.

If EL2 is not implemented, reads of this register return the value of the MIDR_EL1, and writes to the register are ignored.

RW fields in this register reset to architecturally UNKNOWN values.

VPIDR_EL2 resets to the value of MIDR_EL1.

Bit fields and details not provided in this description are architecturally defined. See the *ARM® Architecture Reference Manual ARMv8, for ARMv8-A architecture profile*.

B2.101 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 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 B2-91 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, [13:0]

RES0 Reserved.

VSESR_EL2 bit assignments when EL1 is using AArch64

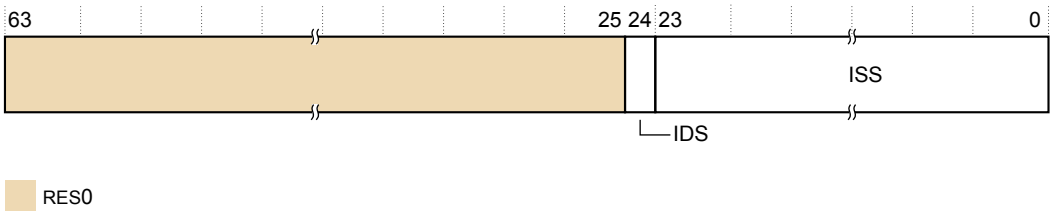


Figure B2-92 VSESR_EL2 bit assignments when EL1 is using AArch64

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 *B1.84 VDFSR, Virtual SError Exception Syndrome Register* on page B1-259.

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.

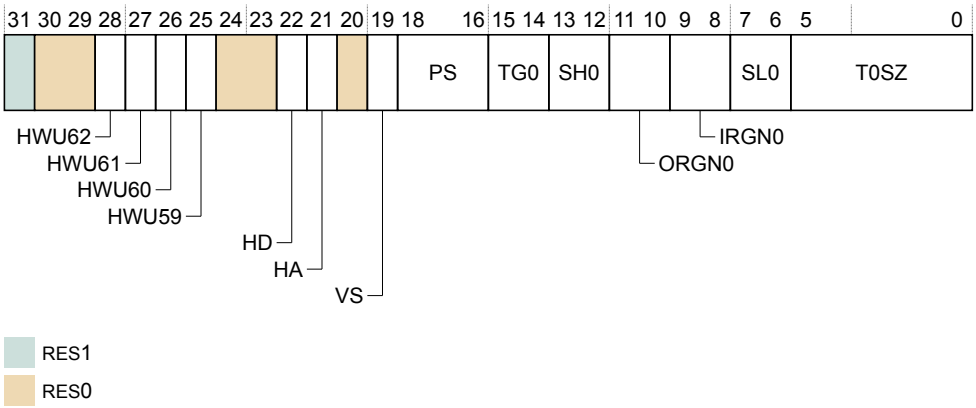


Figure B2-93 VTCR_EL2 bit assignments

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 *B1.88 VTCR, Virtualization Translation Control Register* on page B1-265.

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 modes other than Hyp mode.

Bit field descriptions

VTTBR_EL2 is a 64-bit register, and is part of:

- The Virtualization registers functional group.
- The Virtual memory control registers functional group.

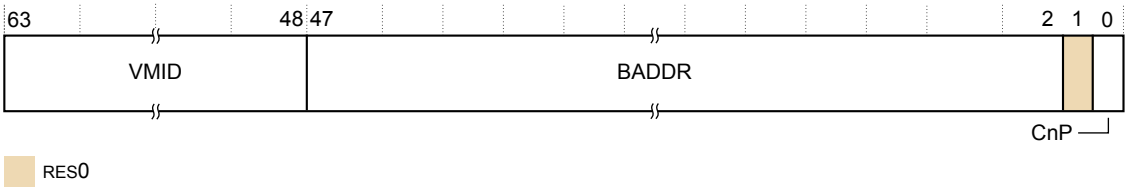


Figure B2-94 VTTBR_EL2 bit assignments

CnP, [0]

Common not Private. The reset value is:

0 CnP is not supported.

Configurations

VTTBR_EL2 is architecturally mapped to AArch32 register VTTBR. See *B1.89 VTTBR, Virtualization Translation Table Base Register* on page B1-267.

Used in conjunction with the VTCR.

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 that are 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 B3-436	*B1.29 ERXADDR, Selected Error Record Address Register* on page B1-169.
			B1.30 ERXADDR2, Selected Error Record Address Register 2 on page B1-170.
			B2.37 ERXADDR_EL1, Selected Error Record Address Register, EL1 on page B2-336
ERR0CTLR	64	*B3.3 ERR0CTLR, Error Record Control Register* on page B3-437	*B1.31 ERXCTLR, Selected Error Record Control Register* on page B1-171.
			B1.32 ERXCTLR2, Selected Error Record Control Register 2 on page B1-172.
			B2.38 ERXCTLR_EL1, Selected Error Record Control Register, EL1 on page B2-337
ERR0FR	64	*B3.4 ERR0FR, Error Record Feature Register* on page B3-439	*B1.33 ERXFR, Selected Error Record Feature Register* on page B1-173.
			B1.34 ERXFR2, Selected Error Record Feature Register 2 on page B1-174.
			B2.39 ERXFR_EL1, Selected Error Record Feature Register, EL1 on page B2-338
ERR0MISC0	64	*B3.5 ERR0MISC0, Error Record Miscellaneous Register 0* on page B3-441	*B1.35 ERXMISC0, Selected Error Miscellaneous Register 0* on page B1-175.
			B1.36 ERXMISC1, Selected Error Miscellaneous Register 1 on page B1-176.
			B2.40 ERXMISC0_EL1, Selected Error Record Miscellaneous Register 0, EL1 on page B2-339
ERR0MISC1	64	*B3.6 ERR0MISC1, Error Record Miscellaneous Register 1* on page B3-443	*B1.37 ERXMISC2, Selected Error Record Miscellaneous Register 2* on page B1-177 accesses bits [31:0]
			B1.38 ERXMISC3, Selected Error Record Miscellaneous Register 3 on page B1-178 accesses bits [63:32]
			B2.41 ERXMISC1_EL1, Selected Error Record Miscellaneous Register 1, EL1 on page B2-340

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 B3-449	*B1.42 ERXSTATUS, Selected Error Record Primary Status Register* on page B1-184
			B2.45 ERXSTATUS_EL1, Selected Error Record Primary Status Register, EL1 on page B2-346

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 B3-444	*B1.39 ERXPFGCDNR, Selected Error Pseudo Fault Generation Count Down Register* on page B1-179
			B2.42 ERXPFGCDNR_EL1, Selected Error Pseudo Fault Generation Count Down Register, EL1 on page B2-341
ERR0PFGCTLR	32	*B3.8 ERR0PFGCTLR, Error Pseudo Fault Generation Control Register* on page B3-445	*B1.40 ERXPFGCTLR, Selected Error Pseudo Fault Generation Control Register* on page B1-181
			B2.43 ERXPFGCTLR_EL1, Selected Error Pseudo Fault Generation Control Register, EL1 on page B2-343
ERR0PFGFR	32	*B3.9 ERR0PFGFR, Error Pseudo Fault Generation Feature Register* on page B3-447	*B1.41 ERXPFGFR, Selected Pseudo Fault Generation Feature Register* on page B1-183
			B2.44 ERXPFGFR_EL1, Selected Pseudo Fault Generation Feature Register, EL1 on page B2-345

B3.2 ERR0ADDR, Error Record Address Register

This register is unused in the Cortex-A55 core and marked as RES0.

Configurations

ERR0ADDR resets to `0x0000000000000000`.

This register is accessible from the following registers when ERRSELR.SEL==0:

- [31:0]: *B1.29 ERXADDR, Selected Error Record Address Register* on page B1-169.
- [63:32]: *B1.30 ERXADDR2, Selected Error Record Address Register 2* on page B1-170.
- *B2.37 ERXADDR_EL1, Selected Error Record Address Register, EL1* on page B2-336.

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.

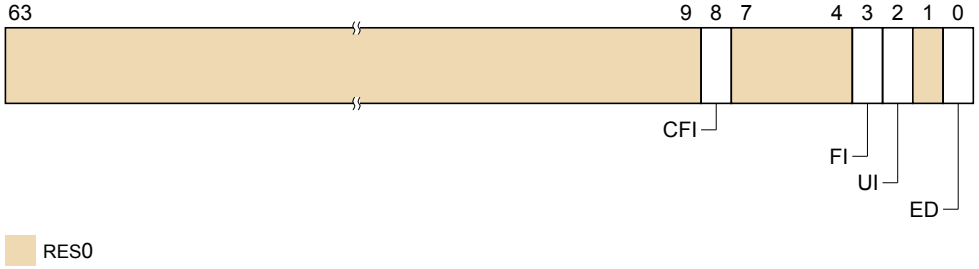


Figure B3-1 ERR0CTLR bit assignments

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. If the node does not support this control, this bit is RES0.

Note

This control 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]

Enable error detection. When disabled, error detection and correction is disabled on reads. Error correction codes are still written for writes. The possible values are:

`0` Error detection and correction disabled.

`1` Error detection and correction enabled.

Note

The bit is set to 0 on Cold reset, meaning errors are not detected or corrected from Cold reset. This allows boot software to initialize the core without signaling errors. When the node is initialized, software can enable error detection.

Configurations

ERR0CTLR resets to `0x0000000000000000`.

This register is accessible from the following registers when ERRSELR.SEL==0:

- [31:0]: *B1.31 ERXCTLR, Selected Error Record Control Register* on page B1-171
- [63:32]: *B1.32 ERXCTLR2, Selected Error Record Control Register 2* on page B1-172.
- *B2.38 ERXCTLR_EL1, Selected Error Record Control Register, EL1* on page B2-337.

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.

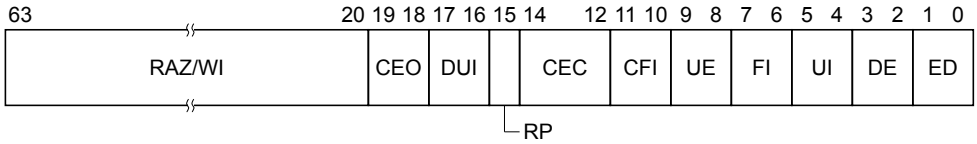


Figure B3-2 ERR0FR bit assignments

[63:20]

Read-as-zero/Write ignore.

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, or no counter is implemented, 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:

`1` 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.

DE, [3:2]

Defers Errors enable. The value is:

`0b01`

Defers Errors always enabled.

ED, [1:0]

Error detection and correction The value is:

`0b10` Error detection is controllable.

Configurations

ERR0FR is accessible from the following registers when ERRSELR.SEL==0:

- [31:0]: *B1.33 ERXFR, Selected Error Record Feature Register* on page B1-173.
- [63:32]: *B1.34 ERXFR2, Selected Error Record Feature Register 2* on page B1-174.
- *B2.39 ERXFR_EL1, Selected Error Record Feature Register, EL1* on page B2-338.

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.

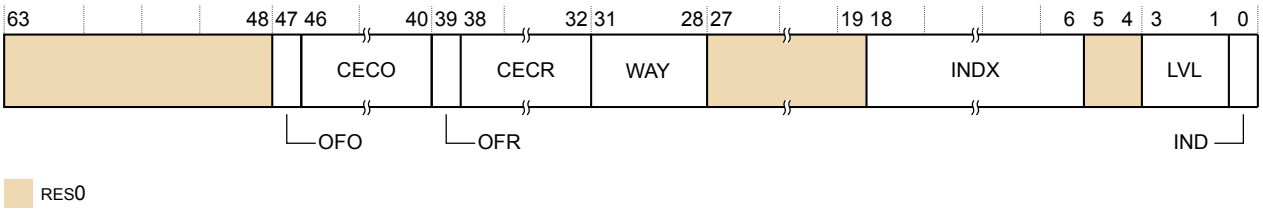


Figure B3-3 ERR0MISC0 bit assignments

RES0, [63:47]

RES0 Reserved.

OFO, [47]

Other Error Count Overflow.

Set when the other error count field overflows. The fault handling interrupt will be asserted when this bit is set and the corrected fault handling interrupt is enabled.

CECO, [46:40]

Other Error Count.

This field is incremented on any corrected memory error that does not match the location (set/way/level/cache/etc) information in this register.

OFR, [39]

Repeat Error Count Overflow.

Set when the repeat error count field overflows. The fault handling interrupt will be asserted when this bit is set and the corrected fault handling interrupt is enabled.

CECR, [38:32]

Repeat Error Count.

This field is incremented on any corrected memory error that exactly matches the location (set/way/level/cache/etc) information in this register.

WAY, [31:28]

Indicates the way that contained the error.

- For all RAMs in the core, only bits [31:30] are used.
- For the L1 instruction cache RAMs, this indicates the RAM bank rather than the way.

RES0, [27:19]

RES0 Reserved.

INDX, [18:6]

Indicates the index that contained the error.

Upper bits of the index are unused depending on the cache size.

RES0, [5:4]

RES0 Reserved.

LVL, [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 `0x0000000000000000`.

This register is accessible from the following registers when ERRSELR.SEL==0:

- [31:0]: *B1.35 ERXMISC0, Selected Error Miscellaneous Register 0* on page B1-175.
- [63:32]: *B1.36 ERXMISC1, Selected Error Miscellaneous Register 1* on page B1-176.
- *B2.40 ERXMISC0_EL1, Selected Error Record Miscellaneous Register 0, EL1* on page B2-339.

B3.6 ERR0MISC1, Error Record Miscellaneous Register 1

This register is unused in the Cortex-A55 core and marked as RES0.

Configurations

ERR0MISC1 is accessible from the following registers when ERRSELR.SEL==0:

- [31:0]: *B1.37 ERXMISC2, Selected Error Record Miscellaneous Register 2* on page B1-177.
- [63:32]: *B1.38 ERXMISC3, Selected Error Record Miscellaneous Register 3* on page B1-178.
- *B2.41 ERXMISC1_EL1, Selected Error Record Miscellaneous Register 1, EL1* on page B2-340.

B3.7 ERR0PFGCDNR, Error Pseudo Fault Generation Count Down Register

ERR0PFGCDNR is the Cortex-A55 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.

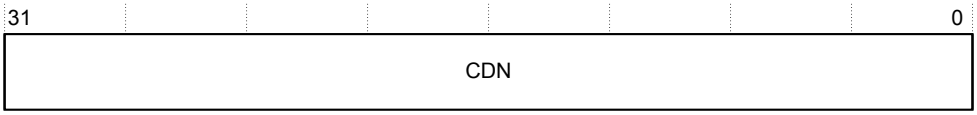


Figure B3-4 ERR0PFGCDNR bit assignments

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 `0x00000000`.

ERR0PFGCDNR is accessible from the following registers when ERRSELR.SEL==0:

- *B1.39 ERXPFGCDNR, Selected Error Pseudo Fault Generation Count Down Register* on page B1-179.
- *B2.42 ERXPFGCDNR_EL1, Selected Error Pseudo Fault Generation Count Down Register, EL1* on page B2-341.

B3.8 ERR0PFGCTLR, Error Pseudo Fault Generation Control Register

The ERR0PFGCTLR is the Cortex-A55 node register that enables controlled fault generation.

Bit field descriptions

ERR0PFGCTLR is a 32-bit read/write register.

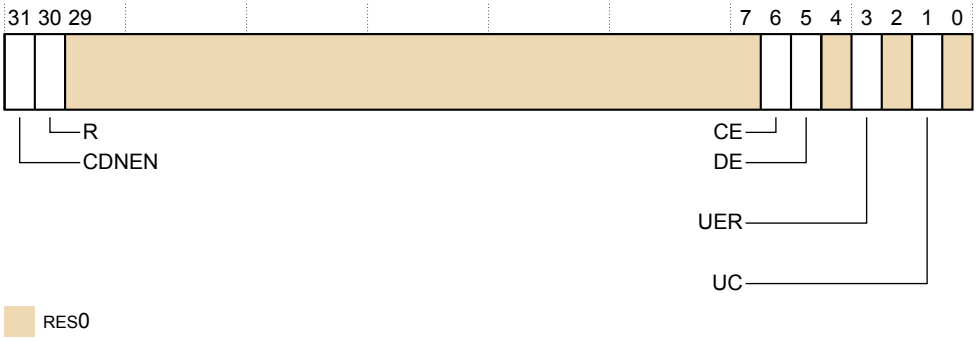


Figure B3-5 ERR0PFGCTLR bit assignments

CDNEN, [31]

Count down enable. This bit controls transfers from the value 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 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 stored in ERR0PFGCDNR and starts counting down again.

RES0, [29:7]

RES0 Reserved.

CE, [6]

Corrected error generation enable. The possible values are:

`0` No corrected error is generated.

`1` A corrected error is generated on the next instruction that could trigger such an error.

DE, [5]

Deferred Error generation enable. The possible values are:

`0` No deferred error is generated.

`1` A deferred error is generated on the next instruction that could trigger such an error.

RES0, [4]

RES0 Reserved.

UER, [3]

Signaled or Recoverable Error generation enable. This bit controls whether a signaled or a recoverable error might be generated. The possible values are:

`0` No signaled or recoverable error will be generated.

`1` A signaled or a recoverable error is generated on the next instruction that could trigger such an error.

RES0, [2]

RES0 Reserved.

UC, [1]

Uncontainable error generation enable. The possible values are:

`0` No uncontainable error is generated.

`1` An uncontainable error is generated on the next instruction that could trigger such an error.

[0]

Reserved, RES0.

Configurations

There are no configuration notes.

ERR0PFGCTLR resets to `0x00000000`.

ERR0PFGCTLR is accessible from the following registers when ERRSELR.SEL==0:

- *B1.40 ERXPFGCTLR, Selected Error Pseudo Fault Generation Control Register* on page B1-181.
- *B2.43 ERXPFGCTLR_EL1, Selected Error Pseudo Fault Generation Control Register, EL1* on page B2-343.

B3.9 ERR0PFGFR, Error Pseudo Fault Generation Feature Register

The ERR0PFGFR is the Cortex-A55 node register that defines which fault generation features are implemented.

Bit field descriptions

ERR0PFGFR is a 32-bit read-only register.

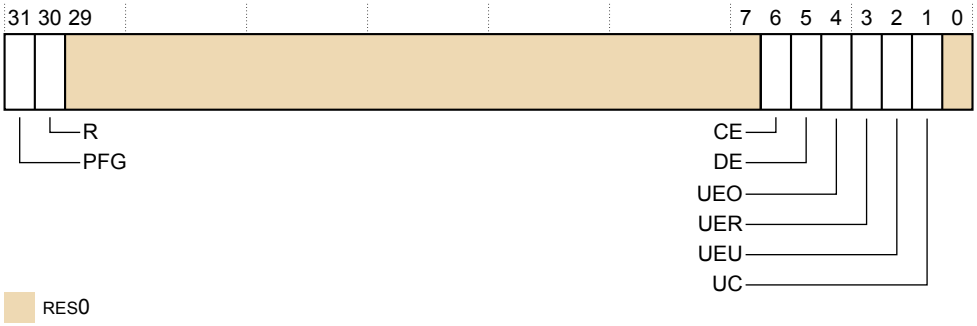


Figure B3-6 ERR0PFGFR bit assignments

PFG, [31]
Pseudo Fault Generation. The possible values are:

`0` The node does not support fault injection.

`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 possible values are:

`0` The node does not support this feature.

`1` This feature is controllable.

[29:7]

Reserved, RES0.

CE, [6]
Corrected Error generation. The possible values are:

`0` The node does not support this feature.

`1` This feature is controllable.

DE, [5]
Deferred Error generation. The possible values are:

`0` The node does not support this feature.

`1` This feature is controllable.

UEO, [4]
Latent or Restartable Error generation. The possible values are:

`0` The node does not support this feature.

UER, [3]
Signaled or Recoverable Error generation. The possible values are:

`0` The node does not support this feature.

`1` This feature is controllable.

UEU, [2]

Unrecoverable Error generation. The possible values are:

`0` The node does not support this feature.

UC, [1]

Uncontainable Error generation. The possible values are:

`0` The node does not support this feature.

`1` This feature is controllable.

[0]

Reserved, RES0.

Configurations

There are no configuration notes.

ERR0PFGFR resets to `0xC000006E`.

ERR0PFGFR is accessible from the following registers when ERRSELR.SEL==0:

- *B1.41 ERXPFGFR, Selected Pseudo Fault Generation Feature Register* on page B1-183.
- *B2.44 ERXPFGFR_EL1, Selected Pseudo Fault Generation Feature Register, EL1* on page B2-345.

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.

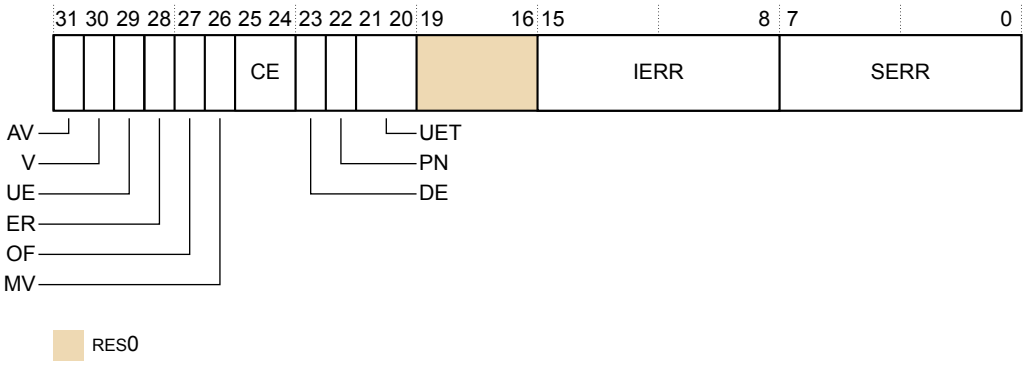


Figure B3-7 ERR0STATUS bit assignments

AV, [31]

Address Valid. The values is:

`0` ERR0ADDR is not valid.

V, [30]

Status Register valid. The possible values are:

`0` ERR0STATUS is not valid.

`1` ERR0STATUS is valid. At least one error has been recorded.

UE, [29]

Uncorrected error. The possible values are:

`0` No error that could not be corrected or deferred has been detected.

`1` At least one error that could not be corrected or deferred has been detected. If error recovery interrupts are enabled, then the interrupt signal is asserted until this bit is cleared.

ER, [28]

Error reported. The possible values are:

`0` No external abort has been reported.

`1` The node has reported an external abort to the master that is in access or making a transaction.

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:
— If a Corrected error counter is implemented, it has not overflowed.
— If no Corrected error counter is implemented, no error status for a Corrected error has been discarded.

`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 recorded by this record.

CE, [25:24]

Corrected error. The possible values are:

`0b00` No corrected errors 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-A55 core cannot distinguish a poisoned value from a corrupted value.

UET, [21:20]

Uncorrected Error Type. The value is:

`0b00` Uncontainable.

RES0, [19:16]

RES0 Reserved.

IERR, [15:8]

Implementation defined error code. The possible values are:

`0x0` No error, or error on other RAMs.

`0x1` Error on L1 dirty RAM.

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:

- *B1.42 ERXSTATUS, Selected Error Record Primary Status Register* on page B1-184.
- *B2.45 ERXSTATUS_EL1, Selected Error Record Primary Status Register, EL1* on page B2-346.

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-A55 core that interfaces with a GIC distributor within the system.

The Cortex-A55 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-A55 core.

Table B4-1 GIC CPU interface system register types supported in the Cortex-A55 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 *B2.50 HCR_EL2, Hypervisor Configuration Register, EL2* on page B2-351.

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 B4-457
ICC_AP1R0	0	12	9	0	RW	*B4.4 ICC_AP1R0, Interrupt Controller Active Priorities Group 1 Register 0* on page B4-458
ICC_BPR0	0	12	8	3	RW	*B4.5 ICC_BPR0, Interrupt Controller Binary Point Register 0* on page B4-459
ICC_BPR1	0	12	12	3	RW	*B4.6 ICC_BPR1, Interrupt Controller Binary Point Register 1* on page B4-460
ICC_CTLR	0	12	12	4	RW	*B4.7 ICC_CTLR, Interrupt Controller Control Register* on page B4-461
ICC_HSRE	4	12	9	5	RW	*B4.8 ICC_HSRE, Interrupt Controller Hyp System Register Enable Register* on page B4-463
ICC_MCTLR	6	12	12	4	RW	*B4.9 ICC_MCTLR, Interrupt Controller Monitor Control Register* on page B4-465
ICC_MSRE	6	12	12	5	RW	*B4.10 ICC_MSRE, Interrupt Controller Monitor System Register Enable Register* on page B4-467
ICC_SRE	0	12	12	5	RW	*B4.11 ICC_SRE, Interrupt Controller System Register Enable Register* on page B4-468

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® Generic Interrupt Controller Architecture Specification*.

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.

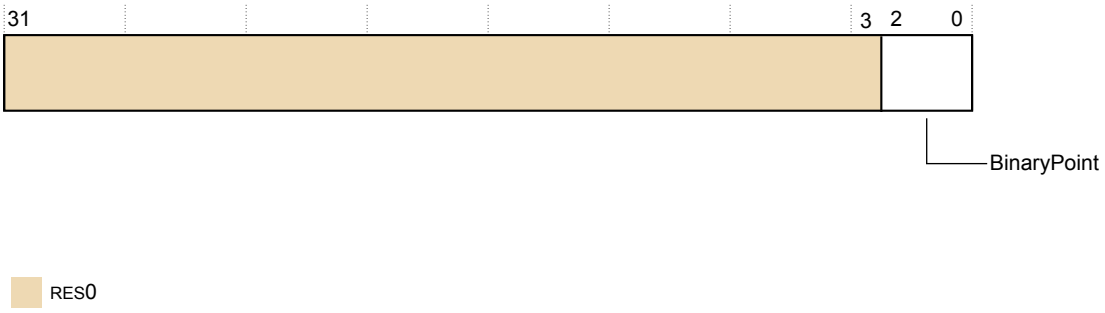


Figure B4-1 ICC_BPR0 bit assignments

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.

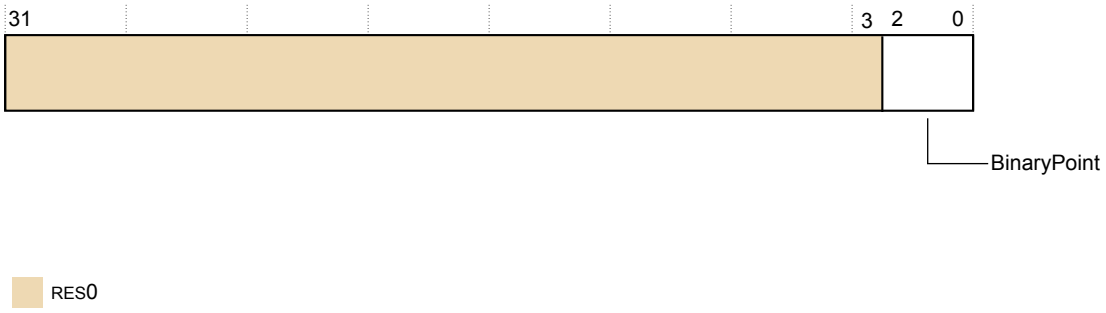


Figure B4-2 ICC_BPR1 bit assignments

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.

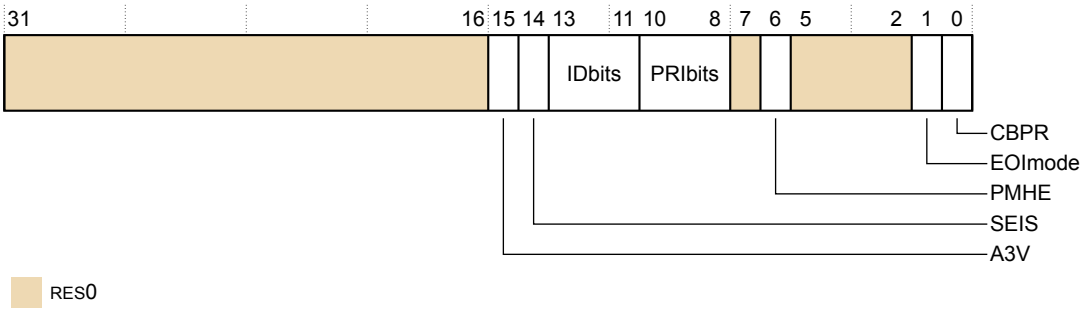


Figure B4-3 ICC_CTLR bit assignments

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.

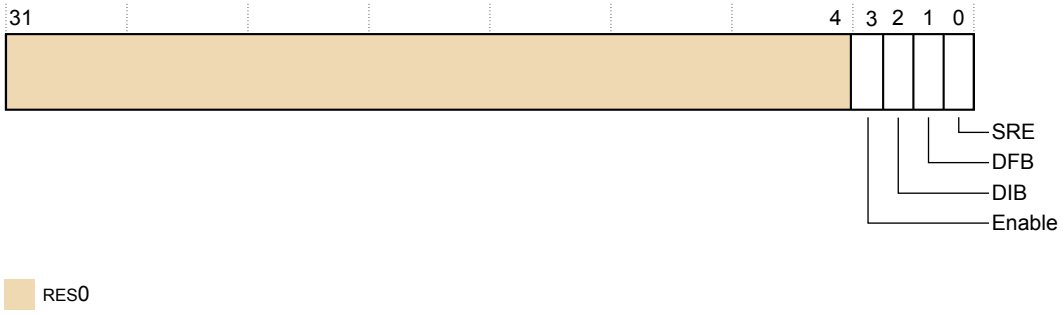


Figure B4-4 ICC_HSRE bit assignments

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.

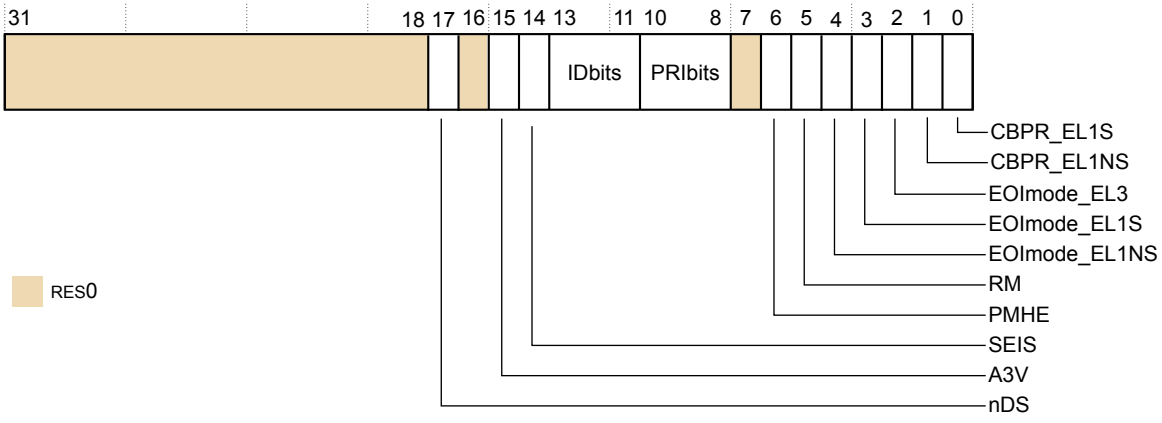


Figure B4-5 ICC_MCTLR bit assignments

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.

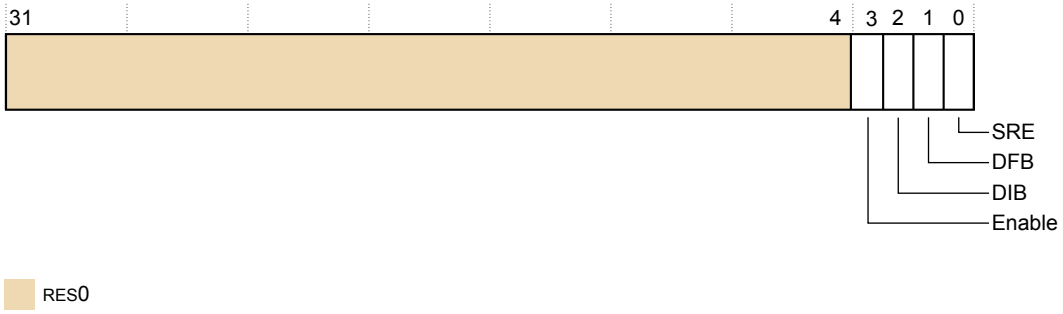


Figure B4-6 ICC_MSRE bit assignments

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.

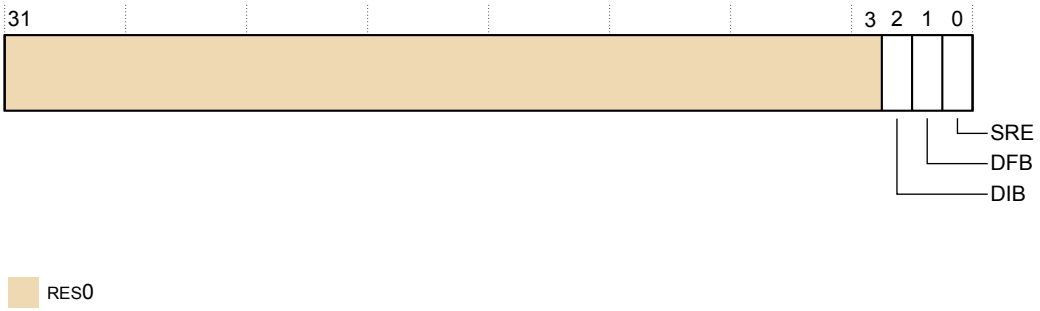


Figure B4-7 ICC_SRE bit assignments

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 B4-470
ICV_AP1R0	0	12	9	0	RW	*B4.14 ICV_AP1R0, Interrupt Controller Virtual Active Priorities Group 1 Register 0* on page B4-471
ICV_BPR0	0	12	8	3	RW	*B4.15 ICV_BPR0, Interrupt Controller Virtual Binary Point Register 0* on page B4-472
ICV_BPR1	0	12	12	3	RW	*B4.16 ICV_BPR1, Interrupt Controller Virtual Binary Point Register 1* on page B4-473
ICV_CTLR	0	12	12	4	RW	*B4.17 ICV_CTLR, Interrupt Controller Virtual Control Register* on page B4-474

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.

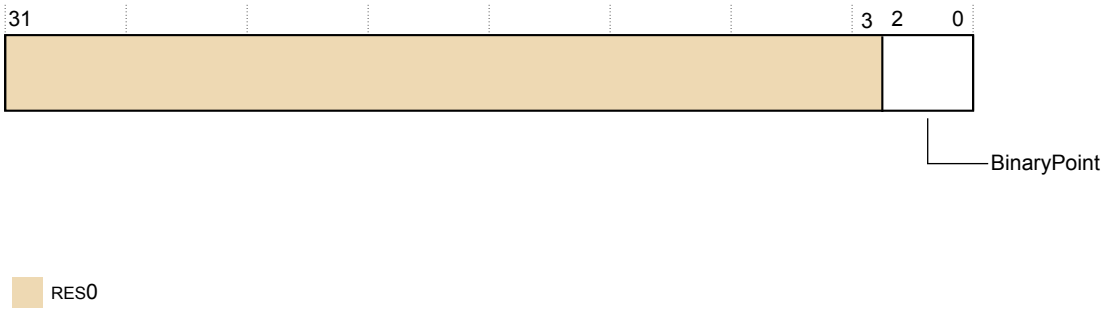


Figure B4-8 ICV_BPR0 bit assignments

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.

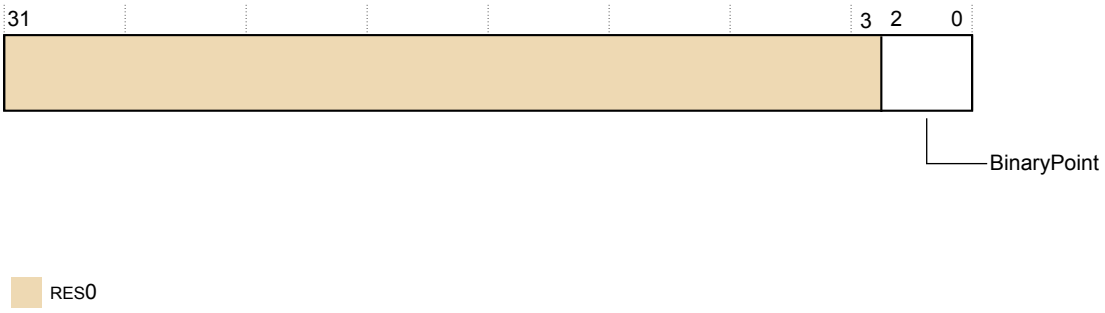


Figure B4-9 ICV_BPR1 bit assignments

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.

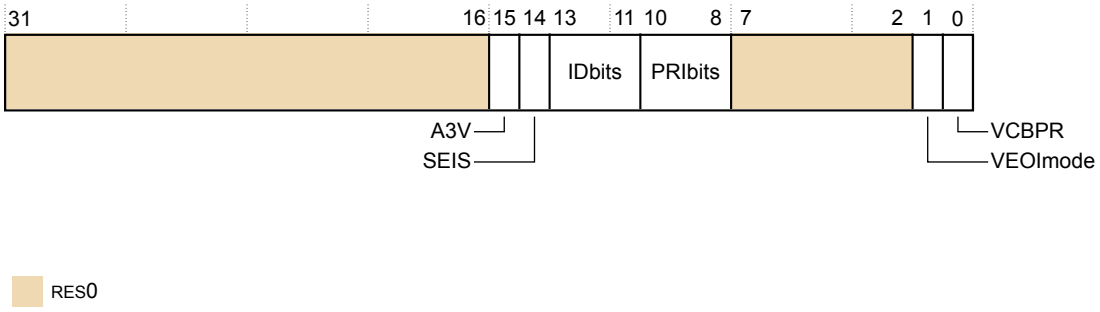


Figure B4-10 ICV_CTLR bit assignments

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 B4-477
ICH_AP1R0	4	12	9	0	RW	*B4.20 ICH_AP1R0, Interrupt Controller Hyp Active Priorities Group 1 Register 0* on page B4-478
ICH_HCR	4	12	11	0	RW	*B4.21 ICH_HCR, Interrupt Controller Hyp Control Register* on page B4-479
ICH_LR0	4	12	12	0	RW	Interrupt Controller List Registers 0-3. The Cortex-A55 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 B4-482
ICH_VMCR	4	12	11	7	RW	*B4.23 ICH_VTR, Interrupt Controller VGIC Type Register* on page B4-484

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.

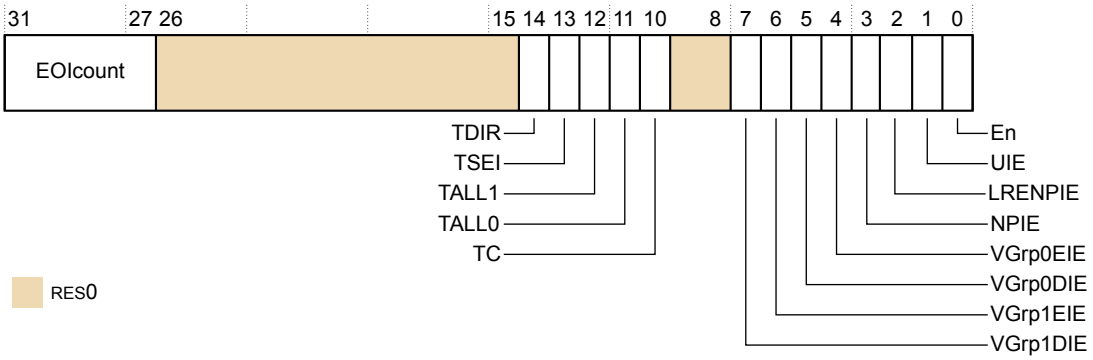


Figure B4-11 ICH_HCR bit assignments

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.

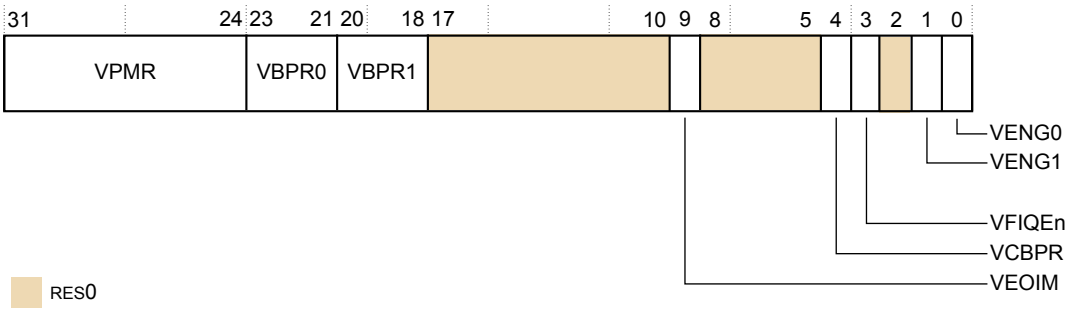


Figure B4-12 ICH_VMCR bit assignments

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.

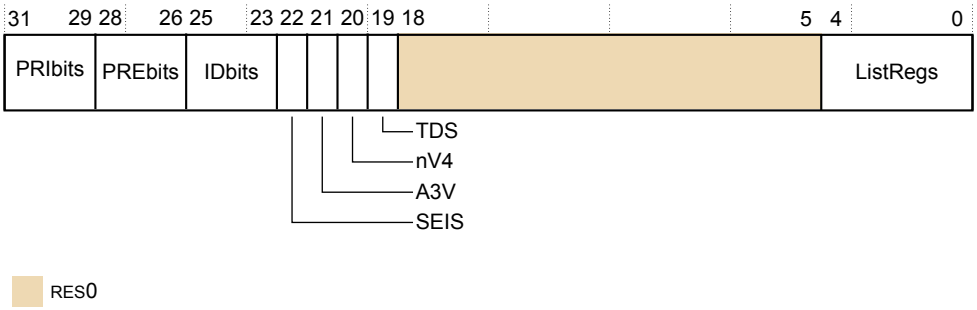


Figure B4-13 ICH_VTR bit assignments

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 B4-487
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 B4-488
ICC_BPR0_EL1	3	0	12	8	3	RW	*B4.27 ICC_BPR0_EL1, Interrupt Controller Binary Point Register 0, EL1* on page B4-489
ICC_BPR1_EL1	3	0	12	12	3	RW	*B4.28 ICC_BPR1_EL1, Interrupt Controller Binary Point Register 1, EL1* on page B4-490
ICC_CTLR_EL1	3	0	12	12	4	RW	*B4.29 ICC_CTLR_EL1, Interrupt Controller Control Register, EL1* on page B4-491
ICC_CTLR_EL3	3	6	12	12	4	RW	*B4.30 ICC_CTLR_EL3, Interrupt Controller Control Register, EL3* on page B4-493
ICC_SRE_EL1	3	0	12	12	5	RW	*B4.31 ICC_SRE_EL1, Interrupt Controller System Register Enable Register, EL1* on page B4-495
ICC_SRE_EL2	3	4	12	9	5	RW	*B4.32 ICC_SRE_EL2, Interrupt Controller System Register Enable register, EL2* on page B4-496
ICC_SRE_EL3	3	6	12	12	5	RW	*B4.33 ICC_SRE_EL3, Interrupt Controller System Register Enable register, EL3* on page B4-498

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-A55 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-A55 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.

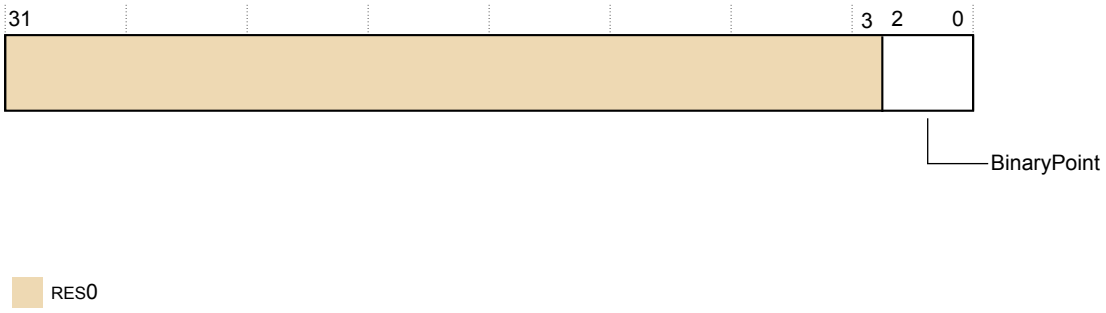


Figure B4-14 ICC_BPR0_EL1 bit assignments

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.

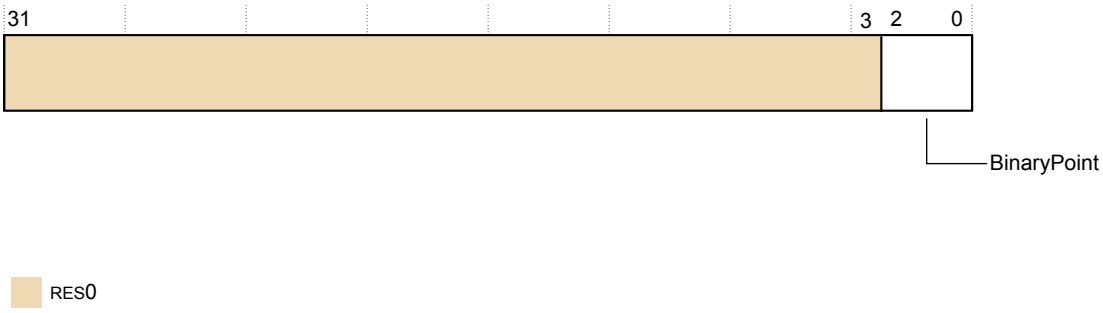


Figure B4-15 ICC_BPR1_EL1 bit assignments

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.

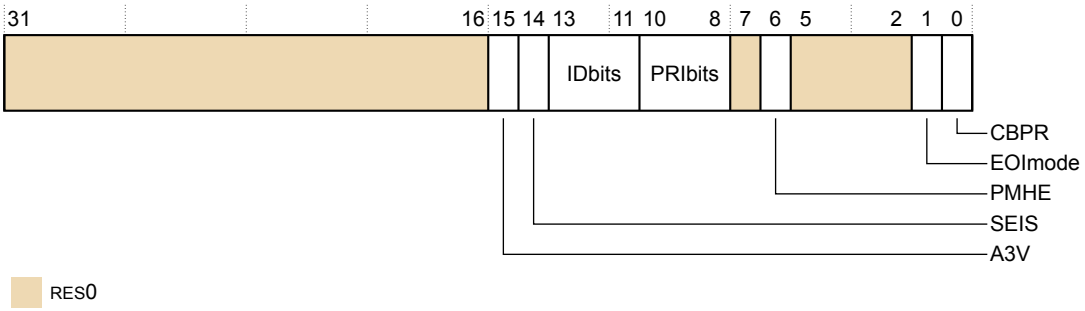


Figure B4-16 ICC_CTLR_EL1 bit assignments

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.

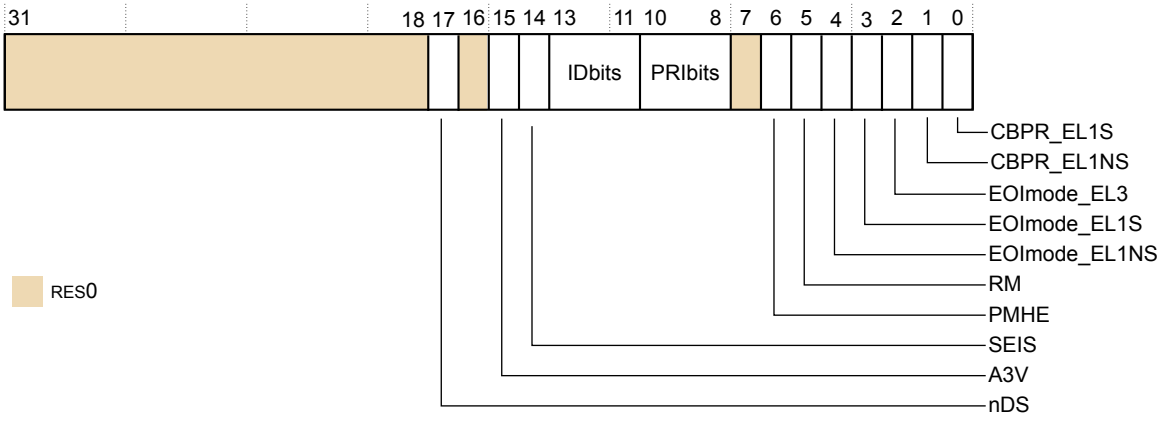


Figure B4-17 ICC_CTLR_EL3 bit assignments

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.

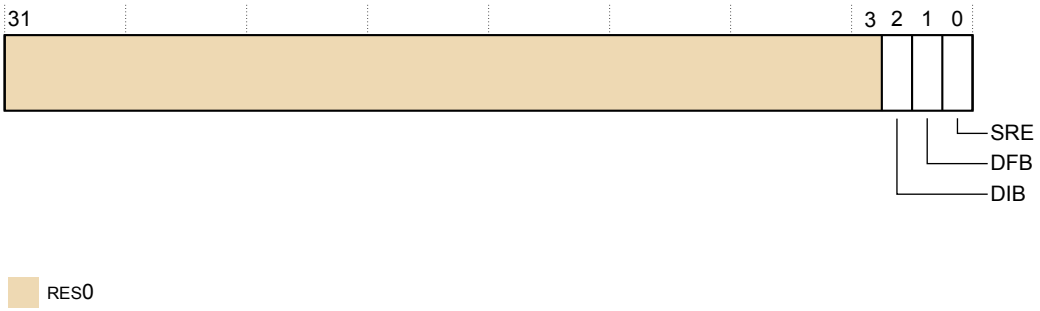


Figure B4-18 ICC_SRE bit assignments

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.

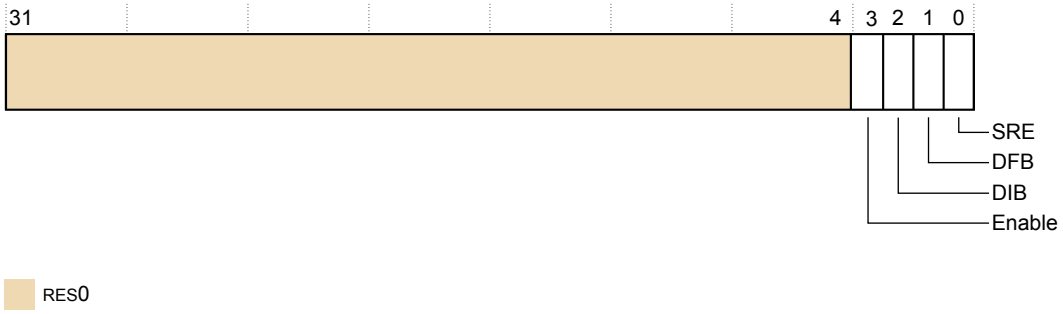


Figure B4-19 ICC_SRE_EL2 bit assignments

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.

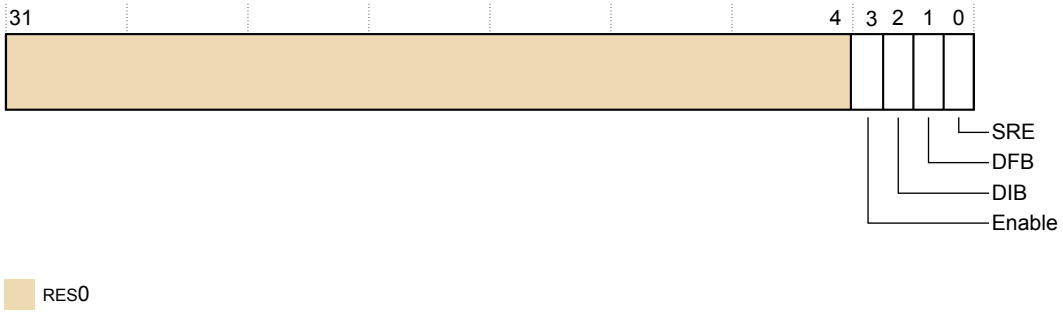


Figure B4-20 ICC_SRE_EL3 bit assignments

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 B4-501
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 B4-502
ICV_BRP0_EL1	3	0	12	8	3	RW	*B4.37 ICV_BPR0_EL1, Interrupt Controller Virtual Binary Point Register 0, EL1* on page B4-503
ICV_BPR1_EL1	3	0	12	12	3	RW	*B4.38 ICV_BPR1_EL1, Interrupt Controller Virtual Binary Point Register 1, EL1* on page B4-504
ICV_CTLR_EL1	3	0	12	12	4	RW	*B4.39 ICV_CTLR_EL1, Interrupt Controller Virtual Control Register, EL1* on page B4-505

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.

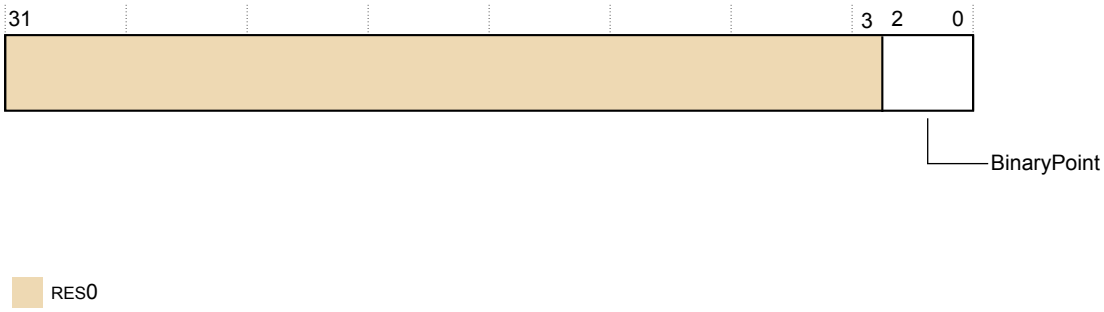


Figure B4-21 ICV_BPR0_EL1 bit assignments

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.

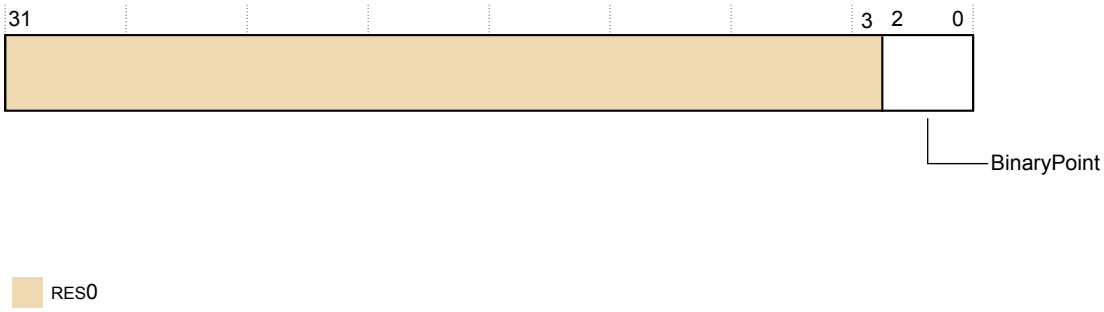


Figure B4-22 ICV_BPR1_EL1 bit assignments

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.

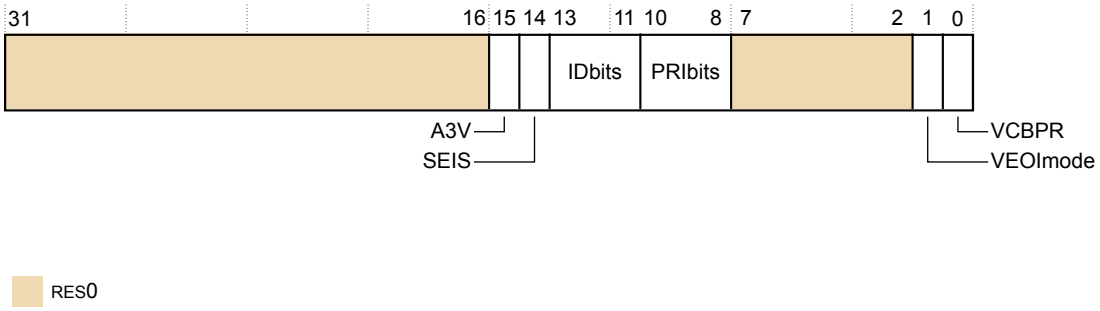


Figure B4-23 ICV_CTLR_EL1 bit assignments

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 B4-508
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 B4-509
ICH_HCR_EL2	3	4	12	11	0	RW	*B4.43 ICH_HCR_EL2, Interrupt Controller Hyp Control Register, EL2* on page B4-510
ICH_VTR_EL2	3	4	12	11	1	RO	*B4.44 ICH_VMCR_EL2, Interrupt Controller Virtual Machine Control Register, EL2* on page B4-513
ICH_VMCR_EL2	3	4	12	11	7	RW	*B4.45 ICH_VTR_EL2, Interrupt Controller VGIC Type Register, EL2* on page B4-515

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.

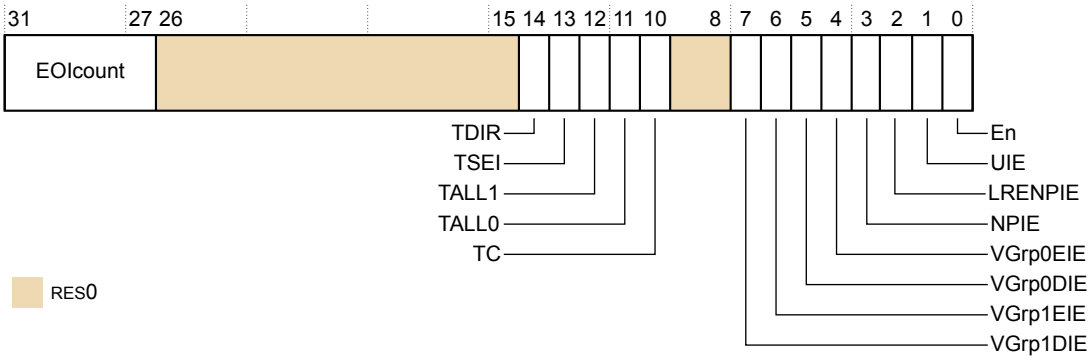


Figure B4-24 ICH_HCR_EL2 bit assignments

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.

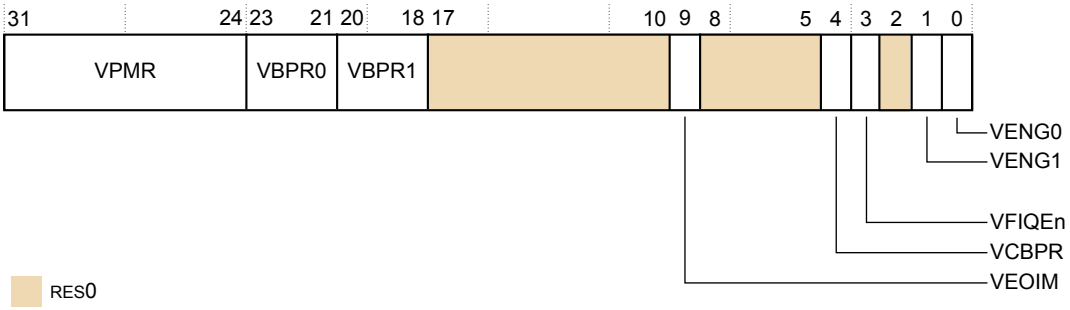


Figure B4-25 ICH_VMCR_EL2 bit assignments

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.

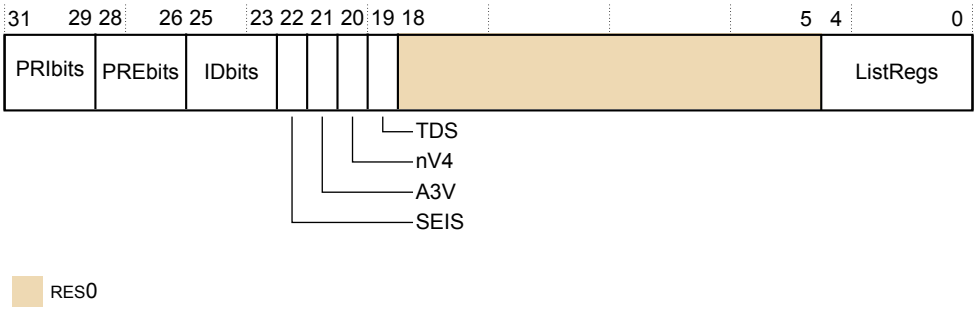


Figure B4-26 ICH_VTR_EL2 bit assignments

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*.

Part C
Debug descriptions

Chapter C1
Debug

This chapter describes the debug features of the core.

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.

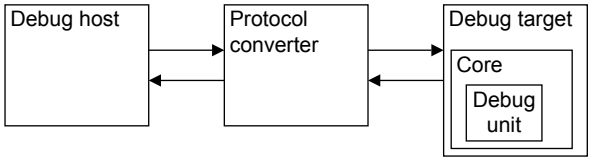


Figure C1-1 External debug system

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-A55 core itself. This way, it does not require expensive interface hardware to connect a second host computer.

C1.2 Debug functional description

This section describes the trace, debug, and test features supported by Cortex-A55. It includes ARMv8-A Debug, CoreSight Debug, and cache Debug.

ARMv8 debug architecture support

The Cortex-A55 core supports the ARMv8-A debug architecture.

The core allows access to the internal debug functionality and registers either through a memory-mapped area on the external AMBA APBv3 slave port, or by using CP14 system coprocessor operations from software running on the core.

The core implements six hardware breakpoints, four watchpoints, and a *Debug Communications Channel* (DCC). Four of the breakpoints match only against virtual address, the other two breakpoints match against either virtual address or context ID. All watchpoints can be linked to either of the virtual address or context-ID matching breakpoints to allow a memory request to be trapped in a given process context.

——— **Note** ———

ARMv7 debug map support

For backwards compatibility, and to reduce the address space required for the debug map, a 4k page-based memory map is also supported.

———

CoreSight debug

The Cortex-A55 core integrates several CoreSight debug related components to aid system debug in conjunction with CoreSight SoC.

These components include:

- Per-core *Embedded Trace Macrocell* (ETM).
- Per-core *Cross Trigger Interface* (CTI).
- *Cross Trigger Matrix* (CTM).
- Debug-over-power-down support.

The following figure shows the Cortex-A55 CoreSight debug components.

——— **Note** ———

The DAP connection is shown for completeness.

———

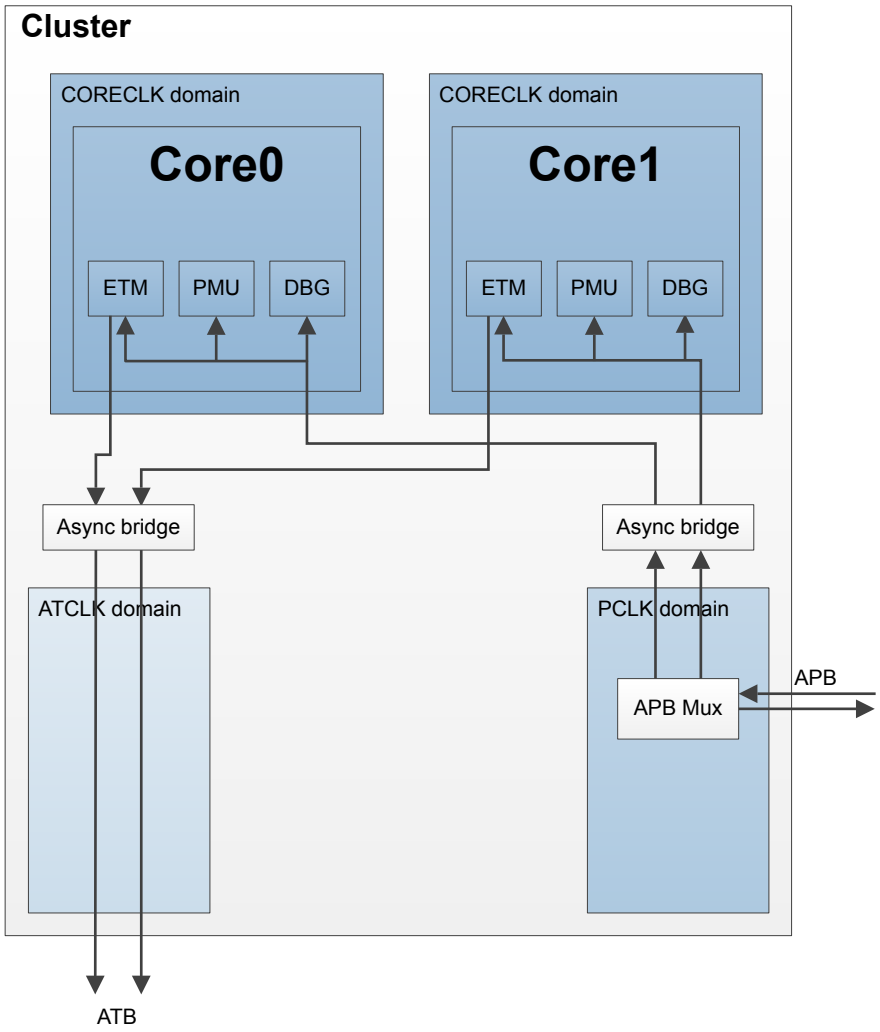


Figure C1-2 Cortex-A55 Debug Components

The debug components are split into two groups. Some components are in the cluster itself and the rest are in a separate block named the DebugBlock. It allows you to put the DebugBlock in a separate power domain and place it physically with other CoreSight logic in the SoC, rather than close to the cluster.

The connection between the cluster and the DebugBlock consists of a pair of APB interfaces, one in each direction. All debug traffic, except the authentication interface, takes place over this interface as read or write APB transactions. It includes register reads, writes, and CTI triggers.

All debug components are controlled through the primary Debug APB interface on the DebugBlock, and form a standard CoreSight interface. Requests on this bus are decoded by the APB decoder before being sent to the appropriate component in the DebugBlock or in the cluster. The per-core CTIs are connected to a CoreSight CTM.

Each core contains an ETM, PMU, and debug component that are accessed using the debug APB bus. This block conforms to the v8 Debug Architecture Specification.

The core supports debug-over-power-down using modules contained in the DebugBlock that mirror key core information such as core ID. These allow the JTAG scan chain connection to be maintained while the core is powered down.

The ETM in each core outputs trace on a 32-bit AMBA 4 ATBv1.1 interface. There is one interface per core.

C1.3 Debug register interfaces

The core implements the ARMv8 Debug architecture and debug events.

They are described in the *ARM® Architecture Reference Manual ARMv8, for ARMv8-A architecture profile*.

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.

C1.3.1 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.

Related references

C1.5 External debug interface on page C1-526.

C1.3.2 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.

nPRESETDBG
This signal initializes the shared debug APB, CTI, and CTM logic. This maps to an External Debug reset that covers the resetting of the external debug interface and has no impact on the core functionality.

C1.3.3 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

C1.4 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.

C1.4.1 Watchpoint debug events

In the Cortex-A55 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.

C1.4.2 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*.

Related references

C1.5 External debug interface on page C1-526.
A3.1 About Clocks, Resets, and Input Synchronization on page A3-42.

C1.5 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
PMU

This chapter describes the *Performance Monitor Unit* (PMU).

It contains the following sections:

C2.1 About the PMU

The Cortex-A55 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-A55 core supports access to the performance monitor registers from the internal system register interface and a memory-mapped interface.

C2.3 External register access permissions to the PMU registers

External access permission to the PMU 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 and memory-mapped interfaces.

Table C2-1 External register conditions

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.
DLK	EDPRSR.DLK is 1	OS Double Lock is locked.
OSLK	OSLSR_EL1.OSLK is 1	OS Lock is locked.
EPMAD	`AllowExternalPMUAccess() == FALSE`	External performance monitors access is disabled. When an error is returned because of an EPMAD condition code, and this is the highest priority error condition, EDPRSR.SPMAD is set to 1. Otherwise SPMAD is unchanged.
Default	-	None of the conditions apply, normal access.

The following table shows an example of external register 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 whose condition is true, the entry gives the register access permission and scanning stops.

Table C2-2 External register condition code example

Off	DLK	OSLK	EPMAD	Default
-	-	-	-	RO

C2.4 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 C2-3 PMU events

Event number	Event mnemonic	PMU event bus (to trace)	Event name
0x00	SW_INCR	-	Instruction architecturally executed, condition code check pass, software increment.
0x01	L1I_CACHE_REFILL	[0]	Level 1 instruction cache refill. This event counts any instruction fetch which misses in the cache. The following instructions are not counted: • Cache maintenance instructions. • Non-cacheable accesses.
0x02	L1I_TLB_REFILL	[1]	Level 1 instruction TLB refill. This event counts any refill of the instruction L1 TLB from the L2 TLB. This includes refills which result in a translation fault. The following instructions are not counted: • TLB maintenance instructions. This event counts regardless of whether the MMU is enabled.
0x03	L1D_CACHE_REFILL	[2]	Level 1 data cache refill. This event counts any load or store operation or pagewalk access which causes data to be read from outside the L1, including accesses which do not allocate into L1. The following instructions are not counted: • Cache maintenance instructions and prefetches. • Stores of an entire cache line, even if they make a coherency request ouside the L1. • Partial cache line writes which do not allocate into the L1 cache. • Non-cacheable accesses. This event counts the sum of L1D_CACHE_REFILL_RD and L1D_CACHE_REFILL_WR.

Table C2-3 PMU events (continued)

Event number	Event mnemonic	PMU event bus (to trace)	Event name
0x04	L1D_CACHE	[3]	Level 1 data cache access. This event counts any load or store operation or pagewalk access which looks up in the L1 data cache. In particular, any access which could count the L1D_CACHE_REFILL event causes this event to count. The following instructions are not counted: • Cache maintenance instructions and prefetches. • Non-cacheable accesses. This event counts the sum of L1D_CACHE_RD and L1D_CACHE_WR.
0x05	L1D_TLB_REFILL	[4]	Level 1 data TLB refill. This event counts any refill of the data L1 TLB from the L2 TLB. This includes refills which result in a translation fault. The following instructions are not counted: • TLB maintenance instructions. This event counts regardless of whether the MMU is enabled.
0x06	LD_RETIRED	[5]	Instruction architecturally executed, condition code check pass, load. This event counts all load and prefetch instructions. This includes the ARMv8.1 atomic instructions, other than the ST* variants.
0x07	ST_RETIRED	[6]	Instruction architecturally executed, condition code check pass, store. This event counts all store instructions and DC ZVA. This includes all the ARMv8.1 atomic instructions. The following instructions are not counted: • Store-Exclusive instructions which fail.
0x08	INST_RETIRED	[7]	Instruction architecturally executed. This event counts all retired instructions, including those that fail their condition check.
0x09	EXC_TAKEN	[8]	Exception taken.
0x0A	EXC_RETURN	[9]	Instruction architecturally executed, condition code check pass, exception return.
0x0B	CID_WRITE_RETIRED	[10]	Instruction architecturally executed, condition code check pass, write to CONTEXTIDR. This event only counts writes to CONTEXTIDR in AArch32, and via the CONTEXTIDR_EL1 mnemonic in AArch64. The following instructions are not counted: • Writes to CONTEXTIDR_EL12 and CONTEXTIDR_EL2.

Table C2-3 PMU events (continued)

Event number	Event mnemonic	PMU event bus (to trace)	Event name
0x0C	PC_WRITE_RETIRED	[11]	Instruction architecturally executed, condition code check pass, software change of the PC. This event counts all branches taken and popped from the branch monitor. This excludes exception entries, debug entries, and CCFAIL branches.
0x0D	BR_IMMED_RETIRED	[12]	Instruction architecturally executed, immediate branch. This event counts all branches decoded as immediate branches, taken or not, and popped from the branch monitor. This excludes exception entries, debug entries, and CCFAIL branches.
0x0E	BR_RETURN_RETIRED	[13]	Instruction architecturally executed, condition code check pass, procedure return.
0x0F	UNALIGNED_LDST_RETIRED	[14]	Instruction architecturally executed, condition code check pass, unaligned load or store.
0x10	BR_MIS_PRED	[15]	Mispredicted or not predicted branch speculatively executed.This event counts any predictable branch instruction which is mispredicted either due to dynamic misprediction or because the MMU is off and the branches are statically predicted not taken.
0x11	CPU_CYCLES	-	Cycle.
0x12	BR_PRED	[16]	Predictable branch speculatively executed. This event counts all predictable branches.
0x13	MEM_ACCESS	[17]	Data memory access.This event counts memory accesses due to load or store instructions. The following instructions are not counted: • Instruction fetches. • Cache maintenance instructions. • Translation table walks or prefetches. This event counts the sum of MEM_ACCESS_RD and MEM_ACCESS_WR.
0x14	L1I_CACHE	[18]	Level 1 instruction cache access. This event counts any instruction fetch which accesses the L1 instruction cache. The following instructions are not counted: • Cache maintenance instructions. • Non-cacheable accesses.

Table C2-3 PMU events (continued)

Event number	Event mnemonic	PMU event bus (to trace)	Event name
0x15	L1D_CACHE_WB	[19]	Level 1 data cache Write-Back. This event counts any write back of data from the L1 data cache to L2 or L3. This counts both victim line evictions and snoops, including cache maintenance operations. The following instructions are not counted: • Invalidations which do not result in data being transferred out of the L1. • Full-line writes which write to L2 without writing L1, such as write-streaming mode.
0x16	L2D_CACHE	[20]	Level 2 data cache access. • If the core is configured with a per-core L2 cache: This event counts any transaction from L1 which looks up in the L2 cache, and any write-back from the L1 to the L2. Snoops from outside the core and cache maintenance operations are not counted. • If the core is not configured with a per-core L2 cache: This event counts the cluster cache event, as defined by L3D_CACHE. • If there is neither a per-core cache nor a cluster cache configured, then this event is not implemented.
0x17	L2D_CACHE_REFILL	[21]	Level 2 data cache refill. • If the core is configured with a per-core L2 cache: This event counts any cacheable transaction from L1 which causes data to be read from outside the core. L2 refills caused by stashes into L2 should not be counted. • If the core is not configured with a per-core L2 cache: This event counts the cluster cache event, as defined by L3D_CACHE_REFILL. • If there is neither a per-core cache nor a cluster cache configured, then this event is not implemented.
0x18	L2D_CACHE_WB	[22]	Level 2 data cache Write-Back. • If the core is configured with a per-core L2 cache: This event counts any write back of data from the L2 cache to outside the core. This includes snoops to the L2 which return data, regardless of whether they cause an invalidation. Invalidations from the L2 which do not write data outside of the core and snoops which return data from the L1 are not counted. • If the core is not configured with a per-core L2 cache, this event is not implemented.

Table C2-3 PMU events (continued)

Event number	Event mnemonic	PMU event bus (to trace)	Event name
0x19	BUS_ACCESS	[23]	Bus access. This event counts for every beat of data transferred over the data channels between the core and the SCU. If both read and write data beats are transferred on a given cycle, this event is counted twice on that cycle. This event counts the sum of BUS_ACCESS_RD and BUS_ACCESS_WR.
0x1A	MEMORY_ERROR	[24]	Local memory error. This event counts any correctable or uncorrectable memory error (ECC or parity) in the protected core RAMs.
0x1B	INT_SPEC	-	Operation speculatively executed. This event duplicates INST_RETIRED.
0x1C	TTBR_WRITE_RETIRED	[25]	Instruction architecturally executed, condition code check pass, write to TTBR.This event only counts writes to TTBR0/TTBR1 in AArch32 and TTBR0_EL1/TTBR1_EL1 in AArch64. The following instructions are not counted: • Accesses to TTBR0_EL12/TTBR1_EL12 or TTBR0_EL2/ TTBR1_EL2.
0x1D	BUS_CYCLES	-	Bus cycles.This event duplicates CPU_CYCLES.
0x1E	CHAIN	-	Odd performance counter chain mode.
0x20	L2D_CACHE_ALLOCATE	[26]	Level 2 data cache allocation without refill. • If the core is configured with a per-core L2 cache: This event counts any full cache line write into the L2 cache which does not cause a linefill, including write-backs from L1 to L2 and full-line writes which do not allocate into L1. • If the core is not configured with a per-core L2 cache: This event counts the cluster cache event, as defined by L3D_CACHE_ALLOCATE. • If there is neither a per-core cache nor a cluster cache configured, this event is not implemented.
0x21	BR_RETIRED	[27]	Instruction architecturally executed, branch. This event counts all branches, taken or not, popped from the branch monitor. This excludes exception entries, debug entries, and CCFAIL branches. In the Cortex-A55 core, an ISB is a branch, and even micro architectural ISBs are counted.
0x22	BR__MIS_PRED_RETIRED	[28]	Instruction architecturally executed, mispredicted branch. This event counts any branch counted by BR_RETIRED which is not correctly predicted and causes a pipeline flush.

Table C2-3 PMU events (continued)

Event number	Event mnemonic	PMU event bus (to trace)	Event name
0x23	STALL_FRONTEND	[29]	No operation issued because of the frontend. The counter counts on any cycle when no operations are issued due to the instruction queue being empty.
0x24	STALL_BACKEND	[30]	No operation issued because of the backend. The counter counts on any cycle when no operations are issued due to a pipeline stall.
0x25	L1D_TLB	[31]	Level 1 data TLB access. This event counts any load or store operation which accesses the data L1 TLB. If both a load and a store are executed on a cycle, this event counts twice. This event counts regardless of whether the MMU is enabled.
0x26	L1I_TLB	[32]	Level 1 instruction TLB access. This event counts any instruction fetch which accesses the instruction L1 TLB. This event counts regardless of whether the MMU is enabled.
0x29	L3D_CACHE_ALLOCATE	[33]	Attributable Level 3 unified cache allocation without refill. • If the core is configured with a per-core L2 cache and the cluster is configured with an L3 cache: This event counts any full cache line write into the L3 cache which does not cause a linefill, including write-backs from L2 to L3 and full-line writes which do not allocate into L2. • If either the core is configured without a per-core L2 or the cluster is configured without an L3 cache, this event is not implemented.
0x2A	L3D_CACHE_REFILL	[34]	Attributable Level 3 unified cache refill. • If the core is configured with a per-core L2 cache and the cluster is configured with an L3 cache: This event counts for any cacheable read transaction returning data from the SCU for which the data source was outside the cluster. Transactions such as ReadUnique are counted here as “read” transactions, even though they can be generated by store instructions. • If either the core is configured without a per-core L2 or the cluster is configured without an L3 cache, this event is not implemented.
0x2B	L3D_CACHE	[35]	Attributable Level 3 unified cache access. • If the core is configured with a per-core L2 cache and the cluster is configured with an L3 cache: This event counts for any cacheable read transaction returning data from the SCU, or for any cacheable write to the SCU. • If either the core is configured without a per-core L2 or the cluster is configured without an L3 cache, this event is not implemented.

Table C2-3 PMU events (continued)

Event number	Event mnemonic	PMU event bus (to trace)	Event name
0x2D	L2D_TLB_REFILL	[36]	Attributable Level 2 unified TLB refill. This event counts on any refill of the L2 TLB, caused by either an instruction or data access. This event does not count if the MMU is disabled.
0x2F	L2D_TLB	[37]	Attributable Level 2 unified TLB access. This event counts on any access to the L2 TLB (caused by a refill of any of the L1 TLBs). This event does not count if the MMU is disabled.
0x34	DTLB_WALK	[39]	Access to data TLB that caused a page table walk. This event counts on any data access which causes L2D_TLB_REFILL to count.
0x35	ITLB_WALK	[40]	Access to instruction TLB that caused a page table walk. This event counts on any instruction access which causes L2D_TLB_REFILL to count.
0x36	LL_CACHE_RD	[41]	Last level cache access, read. • If CPUECTLR.EXTLLC is set: This event counts any cacheable read transaction which returns a data source of "interconnect cache". • If CPUECTLR.EXTLLC is not set: This event is a duplicate of the L*D_CACHE_RD event corresponding to the last level of cache implemented – L3D_CACHE_RD if both per-core L2 and cluster L3 are implemented, L2D_CACHE_RD if only one is implemented, or L1D_CACHE_RD if neither is implemented.
0x37	LL_CACHE_MISS_RD	[42]	Last level cache miss, read. • If CPUECTLR.EXTLLC is set: This event counts any cacheable read transaction which returns a data source of "DRAM", "remote" or "inter-cluster peer". • If CPUECTLR.EXTLLC is not set: This event is a duplicate of the L*D_CACHE_REFILL_RD event corresponding to the last level of cache implemented – L3D_CACHE_REFILL_RD if both per-core L2 and cluster L3 are implemented, L2D_CACHE_REFILL_RD if only one is implemented, or L1D_CACHE_REFILL_RD if neither is implemented.
0x38	REMOTE_ACCESS_RD	[38]	Access to another socket in a multi-socket system, read.This event counts any read transaction which returns a data source of "remote".

Table C2-3 PMU events (continued)

Event number	Event mnemonic	PMU event bus (to trace)	Event name
0x40	L1D_CACHE_RD	-	Level 1 data cache access, read. This event counts any load operation or pagewalk access which looks up in the L1 data cache. In particular, any access which could count the L1D_CACHE_REFILL_RD event causes this event to count. The following instructions are not counted: • Cache maintenance instructions and prefetches. • Non-cacheable accesses.
0x41	L1D_CACHE_WR	-	Level 1 data cache access, write. This event counts any store operation which looks up in the L1 data cache. In particular, any access which could count the L1D_CACHE_REFILL event causes this event to count. The following instructions are not counted: • Cache maintenance instructions and prefetches. • Non-cacheable accesses.
0x42	L1D_CACHE_REFILL_RD	-	Level 1 data cache refill, read. This event counts any load operation or pagewalk access which causes data to be read from outside the L1, including accesses which do not allocate into L1. The following instructions are not counted: • Cache maintenance instructions and prefetches. • Non-cacheable accesses.
0x43	L1D_CACHE_REFILL_WR	-	Level 1 data cache refill, write. This event counts any store operation which causes data to be read from outside the L1, including accesses which do not allocate into L1. The following instructions are not counted: • Cache maintenance instructions and prefetches. • Stores of an entire cache line, even if they make a coherency request ouside the L1. • Partial cache line writes which do not allocate into the L1 cache. • Non-cacheable accesses.
0x44	L1D_CACHE_REFILL_INNER	-	Level 1 data cache refill, inner.This event counts any L1 D-cache linefill (as counted by L1D_CACHE_REFILL) which hits in the L2 cache, L3 cache or another core in the cluster.
0x45	L1D_CACHE_REFILL_OUTER	-	Level 1 data cache refill, outer.This event counts any L1 D-cache linefill (as counted by L1D_CACHE_REFILL) which does not hit in the L2 cache, L3 cache or another core in the cluster, and instead obtains data from outside the cluster

Table C2-3 PMU events (continued)

Event number	Event mnemonic	PMU event bus (to trace)	Event name
0x50	L2D_CACHE_RD	-	Level 2 cache access, read. • If the core is configured with a per-core L2 cache: This event counts any read ransaction from L1 which looks up in the L2 cache. Snoops from outside the core are not counted. • If the core is configured without a per-core L2 cache: This event counts the cluster cache event, as defined by L3D_CACHE_RD. • If there is neither a per-core cache nor a cluster cache configured, this event is not implemented
0x51	L2D_CACHE_WR	-	Level 2 cache access, write. • If the core is configured with a per-core L2 cache: This event counts any write transaction from L1 which looks up in the L2 cache or any write-back from L1 which allocates into the L2 cache. Snoops from outside the core are not counted. • If the core is configured without a per-core L2 cache: This event counts the cluster cache event, as defined by L3D_CACHE_WR. • If there is neither a per-core cache nor a cluster cache configured, this event is not implemented
0x52	L2D_CACHE_REFILL_RD	-	Level 2 cache refill, read. • If the core is configured with a per-core L2 cache: This event counts any cacheable read transaction from L1 which causes data to be read from outside the core. L2 refills caused by stashes into L2 should not be counted. Transactions such as ReadUnique are counted here as “read” transactions, even though they can be generated by store instructions. • If the core is configured without a per-core L2 cache: This event counts the cluster cache event, as defined by L3D_CACHE_REFILL_RD. • If there is neither a per-core cache nor a cluster cache configured, this event is not implemented
0x53	L2D_CACHE_REFILL_WR	-	Level 2 cache refill, write. • If the core is configured with a per-core L2 cache: This event counts any write transaction from L1 which causes data to be read from outside the core. L2 refills caused by stashes into L2 should not be counted. Transactions such as ReadUnique are not counted as write transactions. • If the core is configured without a per-core L2 cache: This event counts the cluster cache event, as defined by L3D_CACHE_REFILL_WR. • If there is neither a per-core cache nor a cluster cache configured, this event is not implemented

Table C2-3 PMU events (continued)

Event number	Event mnemonic	PMU event bus (to trace)	Event name
0x60	BUS_ACCESS_RD	-	Bus access, read. This event counts for every beat of data transferred over the read data channel between the core and the SCU.
0x61	BUS_ACCESS_WR	-	Bus access, write. This event counts for every beat of data transferred over the write data channel between the core and the SCU.
0x66	MEM_ACCESS_RD	-	Data memory access, read.This event counts memory accesses due to load instructions. The following instructions are not counted: • Instruction fetches. • Cache maintenance instructions. • Translation table walks. • Prefetches.
0x67	MEM_ACCESS_WR	-	Data memory access, write.This event counts memory accesses due to store instructions. The following instructions are not counted: • Instruction fetches. • Cache maintenance instructions. • Translation table walks. • Prefetches.
0x70	LD_SPEC	-	Operation speculatively executed, load. This event duplicates LD_RETIRED.
0x71	ST_SPEC	-	Operation speculatively executed, store. This event duplicates ST_RETIRED.
0x72	LDST_SPEC	-	Operation speculatively executed, load or store. This event counts the sum of LD_SPEC and ST_SPEC.
0x73	DP_SPEC	-	Operation speculatively executed, integer data processing. This event counts retired integer data-processing instructions.
0x74	ASE_SPEC	-	Operation speculatively executed, Advanced SIMD instruction. This event counts retired Advanced SIMD instructions.
0x75	VFP_SPEC	-	Operation speculatively executed, floating-point instruction. This event counts retired floating-point instructions.
0x76	PC_WRITE_SPEC	-	Operation speculatively executed, software change of the PC. This event counts retired branch instructions.

Table C2-3 PMU events (continued)

Event number	Event mnemonic	PMU event bus (to trace)	Event name
0x77	CRYPTO_SPEC	-	Operation speculatively executed, Cryptographic instruction. This event counts retired Cryptographic instructions.
0x78	BR_IMMED_SPEC	-	Branch speculatively executed, immediate branch. This event duplicates BR_IMMED_RETIRED.
0x79	BR_RETURN_SPEC	-	Branch speculatively executed, procedure return. This event duplicates BR_RETURN_RETIRED.
0x7A	BR_INDIRECT_SPEC	-	Branch speculatively executed, indirect branch.This event counts retired indirect branch instructions.
0x86	EXC_IRQ	-	Exception taken, IRQ.
0x87	EXC_FIQ	-	Exception taken, FIQ.
0xA0	L3D_CACHE_RD	-	Attributable Level 3 unified cache access, read. This event counts for any cacheable read transaction returning data from the SCU. If either the core is configured without a per-core L2 or the cluster is configured without an L3 cache, this event is not implemented.
0xA2	L3D_CACHE_REFILL_RD	-	Attributable Level 3 unified cache refill, read.This event duplicates L3D_CACHE_REFILL. If either the core is configured without a per-core L2 or the cluster is configured without an L3 cache, this event is not implemented.
0xC0	L3D_CACHE_REFILL_PREFETCH	-	Level 3 cache refill due to prefetch. This event counts any linefills from the hardware prefetcher which cause an allocation into the L3 cache. **Note** It might not be possible to both distinguish hardware vs software prefetches and also which prefetches cause an allocation. If so, only hardware prefetches should be counted, regardless of whether they allocate. If either the core is configured without a per-core L2 or the cluster is configured without an L3 cache, this event is not implemented.

Table C2-3 PMU events (continued)

Event number	Event mnemonic	PMU event bus (to trace)	Event name
0xC1	L2D_CACHE_REFILL_PREFETCH	-	Level 2 cache refill due to prefetch. • If the core is configured with a per-core L2 cache: This event does not count. • If the core is configured without a per-core L2 cache: This event counts the cluster cache event, as defined by L3D_CACHE_REFILL_PREFETCH. • If there is neither a per-core cache nor a cluster cache configured, this event is not implemented
0xC2	L1D_CACHE_REFILL_PREFETCH	-	Level 1 data cache refill due to prefetch. This event counts any linefills from the prefetcher which cause an allocation into the L1 D-cache.
0xC3	L2D_WS_MODE	-	Level 2 cache write streaming mode. This event counts for each cycle where the core is in write-streaming mode and not allocating writes into the L2 cache.
0xC4	L1D_WS_MODE_ENTRY	-	Level 1 data cache entering write streaming mode.This event counts for each entry into write-streaming mode.
0xC5	L1D_WS_MODE	-	Level 1 data cache write streaming mode.This event counts for each cycle where the core is in write-streaming mode and not allocating writes into the L1 D-cache.
0xC6	PREDECODE_ERROR	-	Predecode error.
0xC7	L3D_WS_MODE	-	Level 3 cache write streaming mode.This event counts for each cycle where the core is in write-streaming mode and not allocating writes into the L3 cache.
0xC9	BR_COND_PRED	-	Predicted conditional branch executed.This event counts when any branch which can be predicted by the conditional predictor is retired. This event still counts when branch prediction is disabled due to the MMU being off.
0xCA	BR_INDIRECT_MIS_PRED	-	Indirect branch mis-predicted.This event counts when any indirect branch which can be predicted by the BTAC is retired, and has mis-predicted for either the condition or the address. This event still counts when branch prediction is disabled due to the MMU being off.
0xCB	BR_INDIRECT_ADDR_MIS_PRED	-	Indirect branch mis-predicted due to address mis-compare.This event counts when any indirect branch which can be predicted by the BTAC is retired, was taken and correctly predicted the condition, and has mis-predicted the address. This event still counts when branch prediction is disabled due to the MMU being off.

Table C2-3 PMU events (continued)

Event number	Event mnemonic	PMU event bus (to trace)	Event name
0xCC	BR_COND_MIS_PRED	-	Conditional branch mis-predicted.This event counts when any branch which can be predicted by the conditional predictor is retired, and has mis-predicted the condition. This event still counts when branch prediction is disabled due to the MMU being off. Conditional indirect branches which correctly predicted the condition but mis-predicted on the address do not count this event.
0xCD	BR_INDIRECT_ADDR_PRED	-	Indirect branch with predicted address executed.This event counts when any indirect branch which can be predicted by the BTAC is retired, was taken and correctly predicted the condition. This event still counts when branch prediction is disabled due to the MMU being off.
0xCE	BR_RETURN_ADDR_PRED	-	Procedure return with predicted address executed.This event counts when any procedure return which can be predicted by the CRS is retired, was taken and correctly predicted the condition. This event still counts when branch prediction is disabled due to the MMU being off.
0xCF	BR_RETURN_ADDR_MIS_PRED	-	Procedure return mis-predicted due to address mis-compare.This event counts when any procedure return which can be predicted by the CRS is retired, was taken and correctly predicted the condition, and has mis-predicted the address. This event still counts when branch prediction is disabled due to the MMU being off.
0xD0	L2D_LLWALK_TLB	-	Level 2 TLB last-level walk cache access.This event does not count if the MMU is disabled.
0xD1	L2D_LLWALK_TLB_REFILL	-	Level 2 TLB last-level walk cache refill.This event does not count if the MMU is disabled.
0xD2	L2D_L2WALK_TLB	-	Level 2 TLB level-2 walk cache access.This event counts accesses to the level-2 walk cache where the last-level walk cache has missed. The event only counts when the translation regime of the pagewalk uses level 2 descriptors. This event does not count if the MMU is disabled.
0xD3	L2D_L2WALK_TLB_REFILL	-	Level 2 TLB level-2 walk cache refill.This event does not count if the MMU is disabled.
0xD4	L2D_S2_TLB	-	Level 2 TLB IPA cache access. This event counts on each access to the IPA cache. • If a single pagewalk needs to make multiple accesses to the IPA cache, each access is counted. • If stage 2 translation is disabled, this event does not count.
0xD5	L2D_S2_TLB_REFILL	-	Level 2 TLB IPA cache refill. This event counts on each refill of the IPA cache. • If a single pagewalk needs to make multiple accesses to the IPA cache, each access which causes a refill is counted. • If stage 2 translation is disabled, this event does not count.
0xD6	L2D_CACHE_STASH_DROPPED	-	Level 2 cache stash dropped.This event counts on each stash request received from the interconnect or ACP, that is targeting L2 and gets dropped due to lack of buffer space to hold the request.

Table C2-3 PMU events (continued)

Event number	Event mnemonic	PMU event bus (to trace)	Event name
0xE1	STALL_FRONTEND_CACHE	-	No operation issued due to the frontend, cache miss.This event counts every cycle the DPU IQ is empty and there is an instruction cache miss being processed.
0xE2	STALL_FRONTEND_TLB	-	No operation issued due to the frontend, TLB miss.This event counts every cycle the DPU IQ is empty and there is an instruction L1 TLB miss being processed.
0xE3	STALL_FRONTEND_PDERR	-	No operation issued due to the frontend, pre-decode error.This event counts every cycle the DPU IQ is empty and there is a pre-decode error being processed.
0xE4	STALL_BACKEND_ILOCK	-	No operation issued due to the backend interlock.This event counts every cycle that issue is stalled and there is an interlock. Stall cycles due to a stall in Wr (typically awaiting load data) are excluded.
0xE5	STALL_BACKEND_ILOCK_AGU	-	No operation issued due to the backend, interlock, AGU.This event counts every cycle that issue is stalled and there is an interlock that is due to a load/store instruction waiting for data to calculate the address in the AGU. Stall cycles due to a stall in Wr (typically awaiting load data) are excluded.
0xE6	STALL_BACKEND_ILOCK_FPU	-	No operation issued due to the backend, interlock, FPU.This event counts every cycle that issue is stalled and there is an interlock that is due to an FPU/NEON instruction. Stall cycles due to a stall in the Wr stage (typically awaiting load data) are excluded.
0xE7	STALL_BACKEND_LD	-	No operation issued due to the backend, load.This event counts every cycle there is a stall in the Wr stage due to a load.
0xE8	STALL_BACKEND_ST	-	No operation issued due to the backend, store.This event counts every cycle there is a stall in the Wr stage due to a store.
0xE9	STALL_BACKEND_LD_CACHE	-	No operation issued due to the backend, load, cache miss.This event counts every cycle there is a stall in the Wr stage due to a load which is waiting on data (due to missing the cache or being non-cacheable).
0xEA	STALL_BACKEND_LD_TLB	-	No operation issued due to the backend, load, TLB miss.This event counts every cycle there is a stall in the Wr stage due to a load which has missed in the L1 TLB.
0xEB	STALL_BACKEND_ST_STB	-	No operation issued due to the backend, store, STB full.This event counts every cycle there is a stall in the Wr stage due to a store which is waiting due to the STB being full.
0xEC	STALL_BACKEND_ST_TLB	-	No operation issued due to the backend, store, TLB miss.This event counts every cycle there is a stall in the Wr stage due to a store which has missed in the L1 TLB.

L2 and L3 cache events (L2D_CACHE*, L3D_CACHE*)

The behavior of these events depends on the configuration of the core.

If the private L2 cache is present, the L2D_CACHE* events count the activity in the private L2 cache, and the L3D_CACHE* events count the activity in the DSU L3 cache (if present).

If the private L2 cache is not present but the DSU L3 cache is present, the L2D_CACHE* events count activity in the DSU L3 cache and the L3D_CACHE* events do not count. The L2D_CACHE_WB, L2D_CACHE_WR and L2D_CACHE_REFILL_WR events do not count in this configuration.

If neither the private L2 cache nor the DSU L3 cache are present, neither the L2D_CACHE* or L3D_CACHE* events will count.

Last Level cache events (LL_CACHE_*)

The behavior of these events depends on the configuration of the core and the value of the CPUECTLR.EXTLLC/CPUECTLR_EL1.EXTLLC bit.

If the EXTLLC bit is 0:

These events count activity in the last level of data cache implemented in the core. This is the DSU L3 cache if it is present, else the private L2 cache if it is present, otherwise the L1 data cache.

If the EXTLLC bit is 1:

These events count activity in a last level cache outside the core (if present). These events may not count in all implementations.

C2.5 PMU interrupts

The Cortex-A55 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.

C2.6 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
ETM

This chapter describes the *Embedded Trace Macrocell* (ETM) for the Cortex-A55 core.

It contains the following sections:

C3.1 About the ETM

This module performs real-time instruction flow tracing that complies with the ETM architecture, ETMv4.2. As a CoreSight component, it is part of the ARM real-time debug solution.

C3.2 ETM trace unit generation options and resources

The following table shows the trace generation options implemented in the Cortex-A55 ETM trace unit.

Table C3-1 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	1
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-A55 ETM trace unit.

Table C3-2 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	47, 4 CTI + 43 PMU
Number of counters implemented	2
Reduced function counter implemented	Not implemented

Table C3-2 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

C3.3 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.

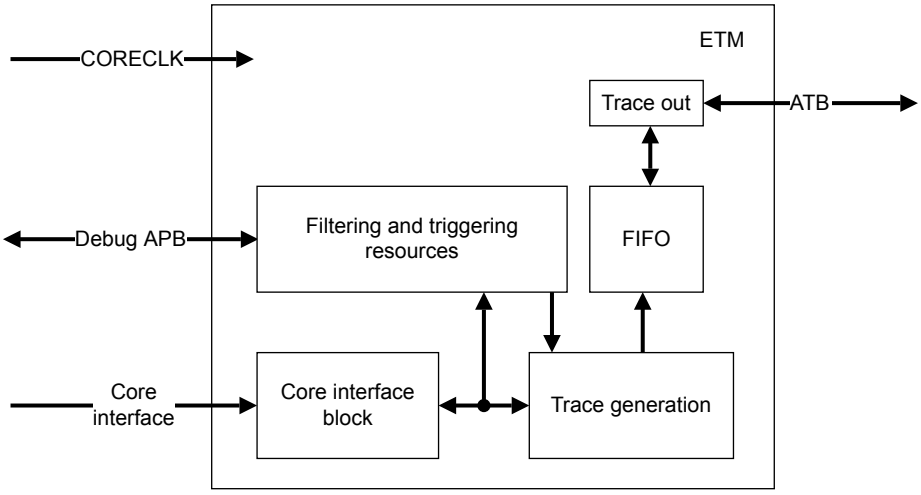


Figure C3-1 ETM functional blocks

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.

C3.4 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.

C3.5 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.

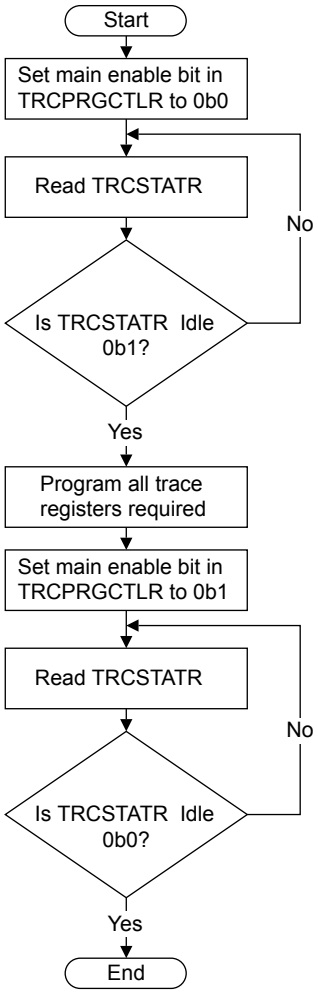


Figure C3-2 Programming ETM trace unit registers

C3.6 ETM trace unit register interfaces

The Cortex-A55 core supports only memory-mapped interface to trace registers.

See the *ARM® ETM Architecture Specification, ETMv4* for information on the behaviors on register accesses for different trace unit states and the different access mechanisms.

Related references

C1.5 External debug interface on page C1-526.

C3.7 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-A55 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 *C2.4 PMU events* on page C2-531 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.

Related references

Chapter C2 PMU on page C2-527.

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*. See the *D3.1 Memory-mapped debug register summary* on page D3-588 for a complete list of registers accessible from the internal memory-mapped interface or the external debug interface.

Table D1-1 AArch32 debug register summary

CRn	Op2	CRm	Op1	Name	Type	Reset	Description
c0	0	c0	0	DBGDIDR	RO	`0x3518D000`	*D1.5 DBGDIDR, Debug ID Register* on page D1-570
c0	0	c1	0	DBGDSCRint	RO	`0x00030000`	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[c]	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	`0x00030000`	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`[d]	Debug Breakpoint Value Register 0
c0	4	c1	0	DBGBVR1	RW	`XXXXXXXX`[d]	Debug Breakpoint Value Register 1
c0	4	c2	0	DBGBVR2	RW	`XXXXXXXX`[d]	Debug Breakpoint Value Register 2
c0	4	c3	0	DBGBVR3	RW	`XXXXXXXX`[d]	Debug Breakpoint Value Register 3
c0	4	c4	0	DBGBVR4	RW	`XXXXXXXX`[d]	Debug Breakpoint Value Register 4
c0	4	c5	0	DBGBVR5	RW	`XXXXXXXX`[d]	Debug Breakpoint Value Register 5
c0	5	c0	0	DBGBCR0	RW	`00XXXXXX`[e]	Debug Breakpoint Control Register 0 See *D2.2 DBGBCRn_EL1, Debug Breakpoint Control Registers, EL1* on page D2-578.
c0	5	c1	0	DBGBCR1	RW	`00XXXXXX`[e]	Debug Breakpoint Control Register 1 See *D2.2 DBGBCRn_EL1, Debug Breakpoint Control Registers, EL1* on page D2-578.
c0	5	c2	0	DBGBCR2	RW	`00XXXXXX`[e]	Debug Breakpoint Control Register 2 See *D2.2 DBGBCRn_EL1, Debug Breakpoint Control Registers, EL1* on page D2-578.

Table D1-1 AArch32 debug register summary (continued)

CRn	Op2	CRm	Op1	Name	Type	Reset	Description
c0	5	c3	0	DBGBCR3	RW	00XXXXXX[e]	Debug Breakpoint Control Register 3 See *D2.2 DBGBCRn_EL1, Debug Breakpoint Control Registers, EL1* on page D2-578.
c0	5	c4	0	DBGBCR4	RW	00XXXXXX[f]	Debug Breakpoint Control Register 4 See *D2.2 DBGBCRn_EL1, Debug Breakpoint Control Registers, EL1* on page D2-578.
c0	5	c5	0	DBGBCR5	RW	00XXXXXX[f]	Debug Breakpoint Control Register 5 See *D2.2 DBGBCRn_EL1, Debug Breakpoint Control Registers, EL1* on page D2-578.
c0	6	c0	0	DBGWVR0	RW	XXXXXXXX[d]	Debug Watchpoint Value Register 0
c0	6	c1	0	DBGWVR1	RW	XXXXXXXX[d]	Debug Watchpoint Value Register 1
c0	6	c2	0	DBGWVR2	RW	XXXXXXXX[d]	Debug Watchpoint Value Register 2
c0	6	c3	0	DBGWVR3	RW	XXXXXXXX[d]	Debug Watchpoint Value Register 3
c0	7	c0	0	DBGWCR0	RW	XXXXXXXX[g]	Watchpoint Control Register 0 See *D2.4 DBGWCRn_EL1, Debug Watchpoint Control Registers, EL1* on page D2-582.
c0	7	c1	0	DBGWCR1	RW	XXXXXXXX[g]	Watchpoint Control Register 1 See *D2.4 DBGWCRn_EL1, Debug Watchpoint Control Registers, EL1* on page D2-582.
c0	7	c2	0	DBGWCR2	RW	XXXXXXXX[g]	Watchpoint Control Register 2 See *D2.4 DBGWCRn_EL1, Debug Watchpoint Control Registers, EL1* on page D2-582.
c0	7	c3	0	DBGWCR3	RW	XXXXXXXX[g]	Watchpoint Control Register 3 See *D2.4 DBGWCRn_EL1, Debug Watchpoint Control Registers, EL1* on page D2-582.
c1	0	c0	0	DBGDRAR[31:0]	RO	-	Debug ROM Address Register, RES0
-	-	c1	-	DBGDRAR[63:0]	RO	-	
c1	1	c4	0	DBGBXVR4	RW	XXXXXXXX[h]	Debug Breakpoint Extended Value Register 4
c1	1	c5	0	DBGBXVR5	RW	XXXXXXXX[h]	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	[i]	Debug Power/Reset Control Register
c2	2	c0	0	DBGDSAR[31:0]	RO	-	Debug Self Address Register RES0
-	0	c2	-	DBGDSAR[63:0][j]	RO	-	
c7	7	c0	0	DBGDEVID2	RO	0x00000000	Debug Device ID Register 2, RES0

Table D1-1 AArch32 debug register summary (continued)

CRn	Op2	CRm	Op1	Name	Type	Reset	Description
c7	7	c1	0	DBGDEVID1	RO	0x00000000	*D1.4 DBGDEVID1, Debug Device ID Register 1* on page D1-569
c7	7	c2	0	DBGDEVID	RO	0x00110F10	*D1.3 DBGDEVID, Debug Device ID Register* on page D1-568
c7	6	c8	0	DBGCLAIMSET	RW	0x000000FF	Debug Claim Tag Set Register
c7	6	c9	0	DBGCLAIMCLR	RW	0x00000000	Debug Claim Tag Clear Register
c7	6	c14	0	DBGAUTHSTATUS	RO	0x000000AA[k]	Debug Authentication Status Register

c Previously returned information about the address of the instruction that accessed a watchpoint address. This register is now deprecated and is RES0.
d The actual reset value is {30{1'bx}},2'b0
e The actual reset value is 32'b000000000x0x0x0xxx0000xxxxx00xx0.
f The actual reset value is 32'b00000000xxxx0x0xxx0000xxxxx00xx0.
g The actual reset value is 32'b000xxxxx000x0x0xxxxxxxxxxxxxxx0.
h The actual reset value is 32'hxxxxxxxxx.
i The actual reset value is 31'b0000000000000000000000000000000,EDPRCR.COREPURQ.
j 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.
k The actual reset value is 24'h000000,1'b1,1'b0,1'b1,1'b0,1'b1,1'b0,1'b1,1'b0.

D1.2 DBGBCR, Debug Breakpoint Control Registers

The DBGBCR*n* holds control information for a breakpoint. Each DBGBVR is associated with a DBGBCR to form a *Breakpoint Register Pair* (BRP). DBGBVR*n* is associated with DBGBCR*n* to form BRP*n*. The range of *n* for DBGBCR*n* is 0 to 5.

Bit field descriptions

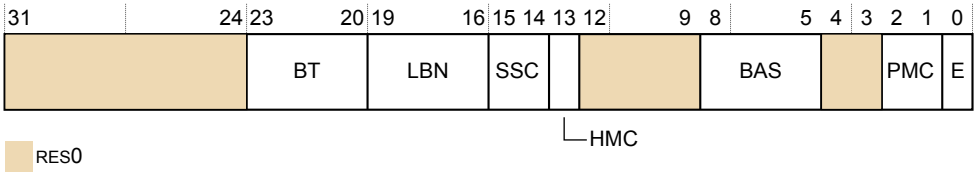


Figure D1-1 DBGBCR

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*, 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.

`0b1000` Unlinked VMID match.

`0b1001` Linked VMID match.

`0b1010` Unlinked VMID + Context ID match.

`0b1011` Linked VMID + Context ID match.

All other values are reserved.

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* is the address of an instruction.

  `0b001` Match context ID. DBGBVR*n*[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* is the address of an instruction to be stepped.

  `0b100` Match VMID. DBGBVR*n*[7:0] is a VMID.

`0b101` Match VMID and context ID. DBGBVR*n*[31:0] is a context ID, and DBGBVR*n*[7:0] is a VMID.

- 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 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 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*.

`0xC` Match the T32 instruction at DBGBVR*n*+2.

`0xF` Match the A64 or A32 instruction at DBGBVR*n*, 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 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*.E is UNKNOWN on reset. A debugger must ensure that DBGBCR*n*.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*.

D1.3 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

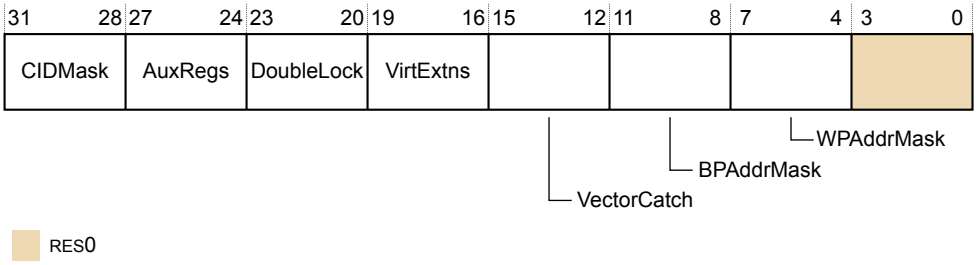


Figure D1-2 DBGDEVID bit assignments

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*.

D1.4 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

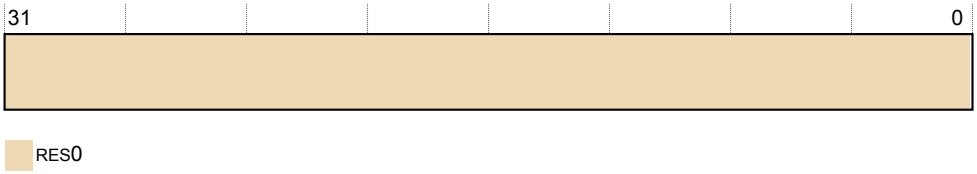


Figure D1-3 DBGDEVID1 bit assignments

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*.

D1.5 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

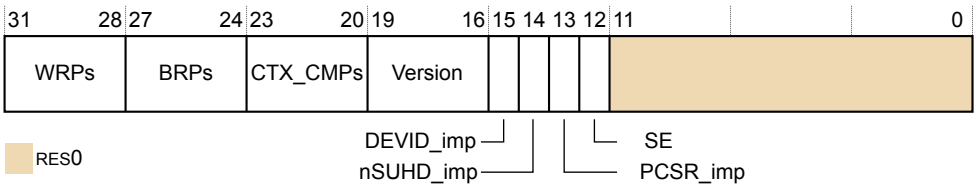


Figure D1-4 DBGDIDR bit assignments

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 ARMv8 Debug architecture.

DEVID_imp, [15]

RAO Reserved.

nSUHD_imp, [14]
Secure User Halting Debug not implemented bit. The value is:

`1` The core does not implement Secure User Halting Debug.

PCSR_imp, [13]

RAZ Reserved.

SE, [12]
EL3 implemented. The value is:

`1` The cluster implements EL3.

RES0, [11:0]

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*.

D1.6 DBGWCR, Debug Watchpoint Control Registers

The DBGWCR*n* holds control information for a watchpoint. Each DBGWCR is associated with a DBGWVR_EL1 to form a *Watchpoint Register Pair* (WRP). DBGWCR*n* is associated with DBGWVR*n*_EL1 to form WRP*n*. The range of *n* for DBGBCR*n* is 0 to 3.

Bit field descriptions

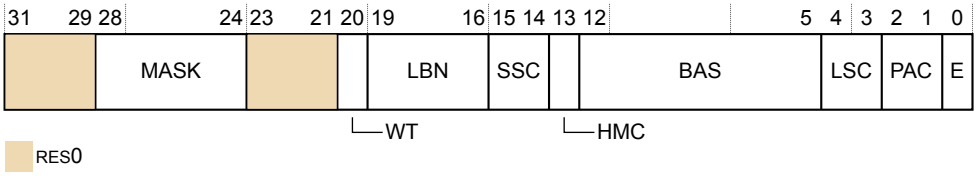


Figure D1-5 DBGWCR

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 PACfields.

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* 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 watchpointed address.

`0b10` Match instructions that store to a watchpointed address.

`0b11` Match instructions that load from or store to a watchpointed 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 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 D3-588 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 D2-578
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 D2-582
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 D2-578
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 D2-582
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 D2-578
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 D2-582
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 D2-578
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 D2-582
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 D2-578
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 D2-578
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

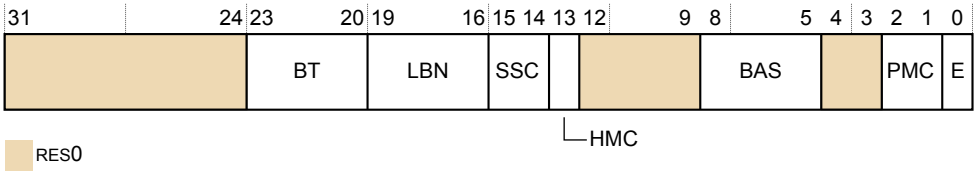


Figure D2-1 DBGBCR*n*_EL1

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.

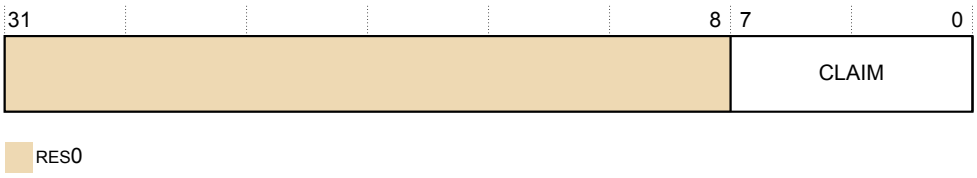


Figure D2-2 DBGCLAIMSET_EL1 bit assignments

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

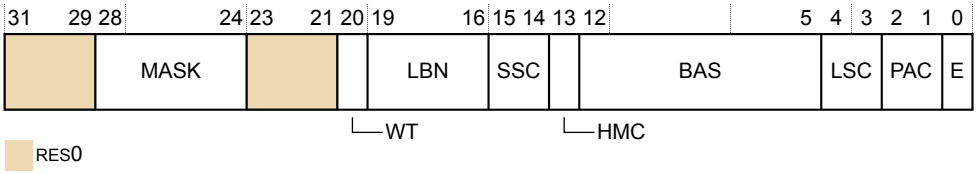


Figure D2-3 DBGWCR_EL1

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*.

D2.5 MDSCR_EL1, Monitor Debug System Control Register, EL1

The MDSCR_EL1 main control register for the debug implementation.

Bit field descriptions

MDSCR_EL1 is a 32-bit register, and is part of the Debug registers functional group.

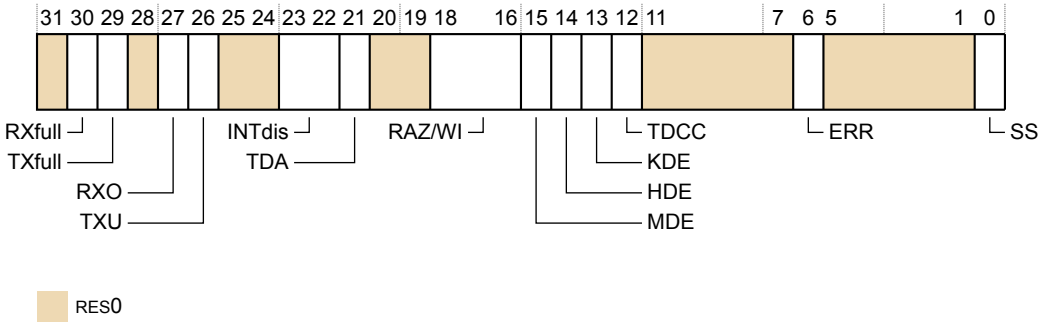


Figure D2-4 MDSCR_EL1 bit assignments

RES0, [31]

RES0 Reserved.

RXfull, [30]

Used for save/restore of EDSCR.RXfull

- When OSLSR_EL1.OSLK == 0 (the OS lock is unlocked), this bit is RO, and software must treat it as UNK/SBZP.
- When OSLSR_EL1.OSLK == 1 (the OS lock is locked), this bit is RW.

TXfull, [29]

Used for save/restore of EDSCR.RXfull

- When OSLSR_EL1.OSLK == 0 (the OS lock is unlocked), this bit is RO, and software must treat it as UNK/SBZP.
- When OSLSR_EL1.OSLK == 1 (the OS lock is locked), this bit is RW.

RES0, [28]

RES0 Reserved.

RXO, [27]

Used for save/restore of EDSCR.RXO.

- When OSLSR_EL1.OSLK == 0 (the OS lock is unlocked), this bit is RO. Software must treat it as UNKNOWN and use an SBZP policy for writes.
- When OSLSR_EL1.OSLK == 1 (the OS lock is locked), this bit is RW.

TXU, [26]

Used for save/restore of EDSCR.TXU.

- When OSLSR_EL1.OSLK == 0 (the OS lock is unlocked), this bit is RO. Software must treat it as UNKNOWN and use an SBZP policy for writes.
- When OSLSR_EL1.OSLK == 1 (the OS lock is locked), this bit is RW.

RES0, [25:24]

RES0 Reserved.

INTdis, [23:22]

Used for save/restore of EDSCR.INTdis.

- When OSLSR_EL1.OSLK == 0 (the OS lock is unlocked), this bit is RO. Software must treat it as UNKNOWN and use an SBZP policy for writes.
- When OSLSR_EL1.OSLK == 1 (the OS lock is locked), this bit is RW.

TDA, [21]

Used for save/restore of EDSCR.TDA.

- When OSLSR_EL1.OSLK == 0 (the OS lock is unlocked), this bit is RO. Software must treat it as UNKNOWN and use an SBZP policy for writes.
- When OSLSR_EL1.OSLK == 1 (the OS lock is locked), this bit is RW.

RES0, [20:19]

RES0 Reserved.

RAZ/WI, [18:16]

Reserved, RAZ/WI. Hardware must implement this as RAZ/WI. Software must not rely on this property as the behavior of reserved values might change in a future revision of the architecture.

MDE, [15]

Monitor debug events. Enable Breakpoint, Watchpoint, and Vector catch debug exceptions.

0 Breakpoint, Watchpoint, and Vector catch debug exceptions disabled.

1 Breakpoint, Watchpoint, and Vector catch debug exceptions enabled.

When this register has an architecturally-defined reset value, this field resets to a value that is architecturally UNKNOWN on Warm reset.

HDE, [14]

Used for save/restore of EDSCR.HDE.

- When OSLSR_EL1.OSLK == 0 (the OS lock is unlocked), this bit is RO. Software must treat it as UNKNOWN and use an SBZP policy for writes.
- When OSLSR_EL1.OSLK == 1 (the OS lock is locked), this bit is RW.

KDE, [13]

Local (kernel) debug enable. If EL_D is using AArch64, enable Software debug events within EL_D. Permitted values are:

0 Software debug events, other than Software breakpoint instructions, disabled within EL_D.

1 Software debug events enabled within EL_D.

RES0 if EL_D is using AArch32.

When this register has an architecturally-defined reset value, this field resets to a value that is architecturally UNKNOWN on Warm reset.

TDCC, [12]

Traps EL0 accesses to the DCC registers to EL1, from both Execution states:

0 EL0 using AArch64:

- EL0 accesses to the MDCCSR_EL0, DBGDTR_EL0, DBGDTRTX_EL0, and DBGDTRRX_EL0 registers are not trapped to EL1.

EL0 using AArch32:

- EL0 accesses to the DBGDSCRint, DBGDTRRXint, DBGDTRTXint, DBGDIDR, DBGDSAR, and DBGDRAR registers are not trapped to EL1.

1 EL0 using AArch64:

- EL0 accesses to the MDCCSR_EL0, DBGDTR_EL0, DBGDTRTX_EL0, and DBGDTRRX_EL0 registers are trapped to EL1.

EL0 using AArch32:

- EL0 accesses to the DBGDSCRint, DBGDTRRXint, DBGDTRTXint, DBGDIDR, DBGDSAR, and DBGDRAR registers are trapped to EL1.

All accesses to these AArch32 registers are trapped, including LDC and STC accesses to DBGDTRTXint and DBGDTRRXint, and MRRC accesses to DBGDSAR and DBGDRAR.

Traps of AArch32 PL0 accesses to the DBGDTRRXint and DBGDTRTXint are ignored in Debug state.

When this register has an architecturally-defined reset value, this field resets to a value that is architecturally UNKNOWN on Warm reset.

RES0, [11:7]

RES0 Reserved.

ERR, [6]

Used for save/restore of EDSCR.ERR.

- When OSLSR_EL1.OSLK == 0 (the OS lock is unlocked), this bit is RO. Software must treat it as UNKNOWN and use an SBZP policy for writes.
- When OSLSR_EL1.OSLK == 1 (the OS lock is locked), this bit is RW.

RES0, [5:1]

RES0 Reserved.

SS, [0]

Software step control bit. If EL_D is using AArch64, enable Software step. Permitted values are:

0 Software step is disabled.

1 Software step is unabled.

RES0 if EL_D is using AArch32.

When this register has an architecturally-defined reset value, this field resets to a value that is architecturally UNKNOWN on Warm reset.

Configurations

AArch64 System register MDSCR_EL1 is architecturally mapped to AArch32 System register DBGDSCRext. See *ARM® Architecture Reference Manual ARMv8, for ARMv8-A architecture profile*.

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 AArch64. Otherwise, on a Warm or Cold reset 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*.

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	*D3.17 EDRCR, External Debug Reserve Control Register* on page D3-609
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 D2-578

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 D2-578
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 D2-578
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 D2-578
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 D2-578
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 D2-578
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 D2-582
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 D2-582
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 D2-582
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 D2-582
0x83C-0xCFC	-	-	-	Reserved
0xD00	MIDR	RO	32	*B2.82 MIDR_EL1, Main ID Register, EL1* on page B2-403
0xD04-0xD1C	-	-	-	Reserved
0xD20	EDPFR[31:0]	RO	64	*B2.58 ID_AA64PFR0_EL1, AArch64 Processor Feature Register 0, EL1* on page B2-363
0xD24	EDPFR[63:32]			
0xD28	EDDFR[31:0]	RO	64	*B2.58 ID_AA64PFR0_EL1, AArch64 Processor Feature Register 0, EL1* on page B2-363
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 D2-581
0xFA4	DBGCLAIMCLR_EL1	RW	32	Debug Claim Tag Clear Register
0xFA8	EDDEVAFF0	RO	32	*B1.74 MPIDR, Multiprocessor Affinity Register* on page B1-234
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 D3-597
0xFC8	EDDEVID	RO	32	*D3.6 EDDEVID, External Debug Device ID Register 0* on page D3-596
0xFCC	EDDEVTYPE	RO	32	External Debug Device Type Register
0xFD0	EDPIDR4	RO	32	*D3.15 EDPIDR4, External Debug Peripheral Identification Register 4* on page D3-607
0xFD4-0xFDC	EDPIDR5-7	RO	32	*D3.16 EDPIDRn, External Debug Peripheral Identification Registers 5-7* on page D3-608

Table D3-1 Memory-mapped debug register summary (continued)

Offset	Name	Type	Width	Description
0xFE0	EDPIDR0	RO	32	*D3.11 EDPIDR0, External Debug Peripheral Identification Register 0* on page D3-603
0xFE4	EDPIDR1	RO	32	*D3.12 EDPIDR1, External Debug Peripheral Identification Register 1* on page D3-604
0xFE8	EDPIDR2	RO	32	*D3.13 EDPIDR2, External Debug Peripheral Identification Register 2* on page D3-605
0xFEC	EDPIDR3	RO	32	*D3.14 EDPIDR3, External Debug Peripheral Identification Register 3* on page D3-606
0xFF0	EDCIDR0	RO	32	*D3.2 EDCIDR0, External Debug Component Identification Register 0* on page D3-592
0xFF4	EDCIDR1	RO	32	*D3.3 EDCIDR1, External Debug Component Identification Register 1* on page D3-593
0xFF8	EDCIDR2	RO	32	*D3.4 EDCIDR2, External Debug Component Identification Register 2* on page D3-594
0xFFC	EDCIDR3	RO	32	*D3.5 EDCIDR3, External Debug Component Identification Register 3* on page D3-595

D3.2 EDCIDR0, External Debug Component Identification Register 0

The EDCIDR0 provides information to identify an external debug component.

Bit field descriptions

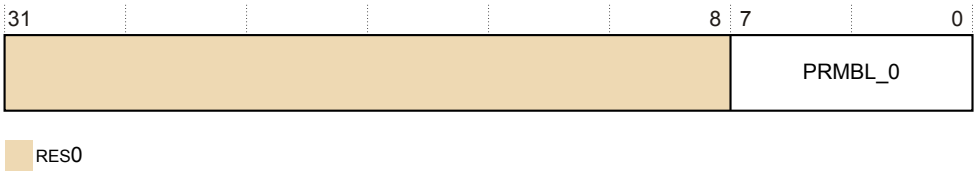


Figure D3-1 EDCIDR0 bit assignments

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

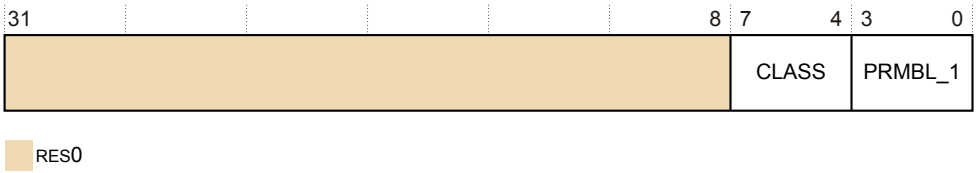


Figure D3-2 EDCIDR1 bit assignments

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

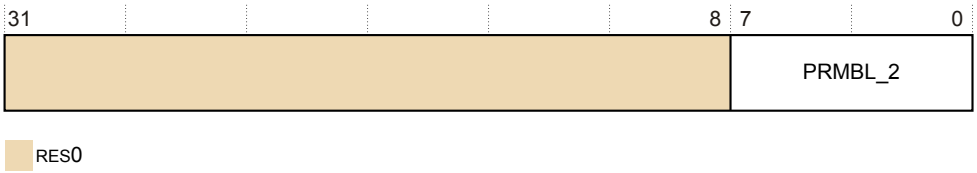


Figure D3-3 EDCIDR2 bit assignments

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

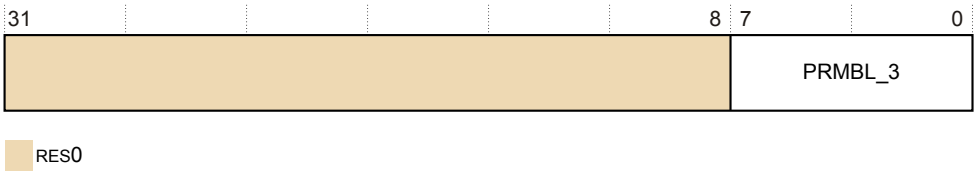


Figure D3-4 EDCIDR3 bit assignments

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

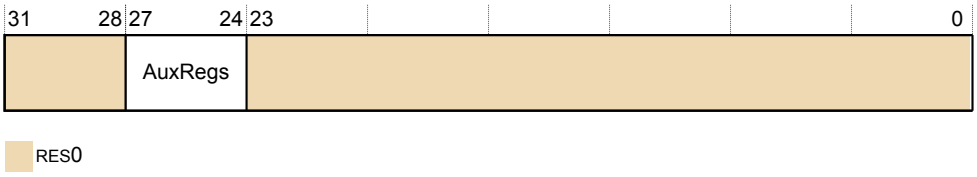


Figure D3-5 EDDEVID bit assignments

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

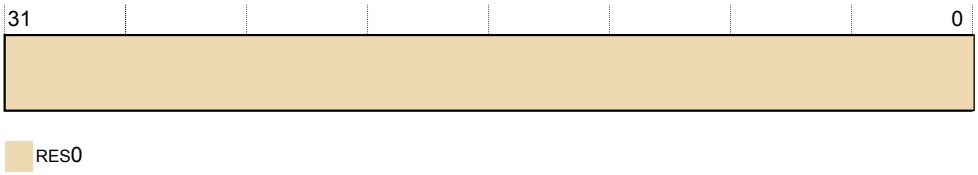


Figure D3-6 EDDEVID1 bit assignments

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 EDDFR, External Debug Feature Register

The EDDFR provides top level information about the debug system in AArch64.

Bit field descriptions

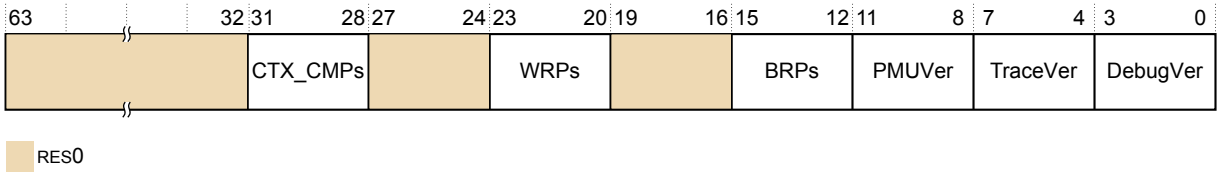


Figure D3-7 EDDFR bit assignments

RES0, [63:32]

RES0 Reserved.

CTX_CMPs, [31:28]

Number of breakpoints that are context-aware, minus 1. These are the highest numbered breakpoints.

RES0, [27:24]

RES0 Reserved.

WRPs, [23:20]

Number of watchpoints, minus 1. The value of `0b0000` is reserved.

RES0, [19:16]

RES0 Reserved.

BRPs, [15:12]

Number of breakpoints, minus 1. The value of `0b0000` is reserved.

PMUVer, [11:8]

Performance Monitors extension version. Indicates whether system register interface to Performance Monitors extension is implemented. Defined values are:

`0x0000` Performance Monitors extension system registers not implemented.

`0x0001` Performance Monitors extension system registers implemented, PMUv3.

`0x11111` IMPLEMENTATION DEFINED form of performance monitors supported, PMUv3 not supported.

All other values are reserved.

TraceVer [7:4]

Trace support. Indicates whether system register interface to a trace macrocell is implemented. Defined values are:

`0x0000` Trace macrocell system registers not implemented.

`0x0001` Trace macrocell system registers implemented.

All other values are reserved.

A value of `0x0000` only indicates that no system register interface to a trace macrocell is implemented. A trace macrocell might nevertheless be implemented without a system register interface.

UNKOWN, [3:0]

UNKNOWN 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*.

EDDFR[31:0] can be accessed through the external debug interface, offset `0xD28`.

EDDFR[63:32] can be accessed through the external debug interface, offset `0xD2C`.

D3.9 EDITCTRL, External Debug Integration Mode Control Register

The EDITCTRL enables the external debug to switch from its default mode into integration mode, where test software can control directly the inputs and outputs of the core, for integration testing or topology detection.

Bit field descriptions

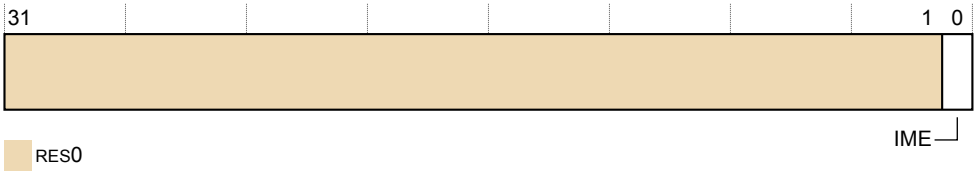


Figure D3-8 EDITCTRL bit assignments

[31:1]

RES0 Reserved.

IME, [0]

Integration Mode Enable.

RES0. The device does not revert to an integration mode to enable integration testing or topology detection.

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 EDITCTRL can be accessed through the external debug interface, offset `0xF00`.

D3.10 EDPFR, External Debug Processor Feature Register

The EDPFR provides additional information about implemented PE features in AArch64.

Bit field descriptions

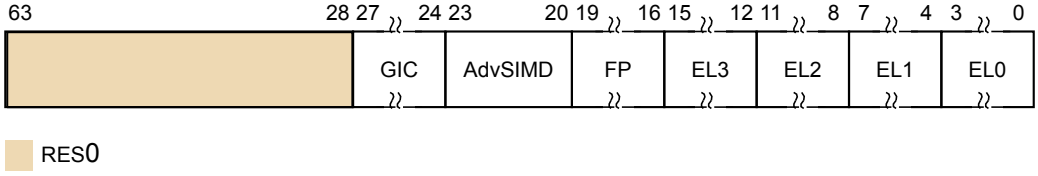


Figure D3-9 EDPFR bit assignments

RES0, [63:28]

RES0 Reserved.

GIC, [27:24]

System register GIC interface. Defined values are:

0x0 No System register interface to the GIC is supported.

0x1 System register interface to the GIC CPU interface is supported.

All other values are reserved.

AdvSIMD, [23:20]

Advanced SIMD. Defined values are:

0x0 Advanced SIMD is implemented.

0xF Advanced SIMD is not implemented.

All other values are reserved.

FP, [19:16]

Floating-point. Defined values are:

0x0 Floating-point is implemented.

0xF Floating-point is not implemented.

All other values are reserved.

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 EL2 in AArch64 or AArch32 state.

EL1 handling, [7:4]

EL1 exception handling. The possible values are:

0x2 Instructions can be executed at EL1 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.

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 EDPFR[31:0] can be accessed through the external debug interface, offset `0xD20`.

The EDPFR[63:32] can be accessed through the external debug interface, offset `0xD24`.

D3.11 EDPIDR0, External Debug Peripheral Identification Register 0

The EDPIDR0 provides information to identify an external debug component.

Bit field descriptions

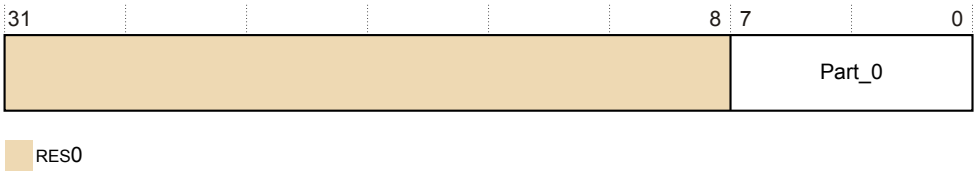


Figure D3-10 EDPIDR0 bit assignments

RES0, [31:8]

RES0 Reserved.

Part_0, [7:0]

`0x05` 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`.

D3.12 EDPIDR1, External Debug Peripheral Identification Register 1

The EDPIDR1 provides information to identify an external debug component.

Bit field descriptions

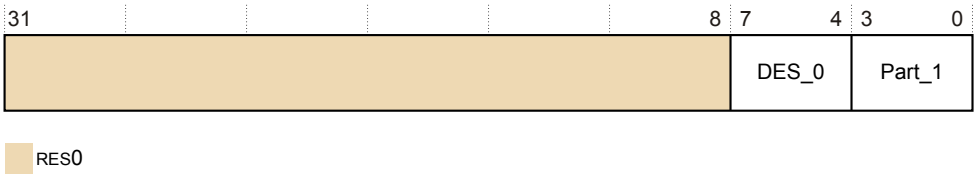


Figure D3-11 EDPIDR1 bit assignments

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`.

D3.13 EDPIDR2, External Debug Peripheral Identification Register 2

The EDPIDR2 provides information to identify an external debug component.

Bit field descriptions

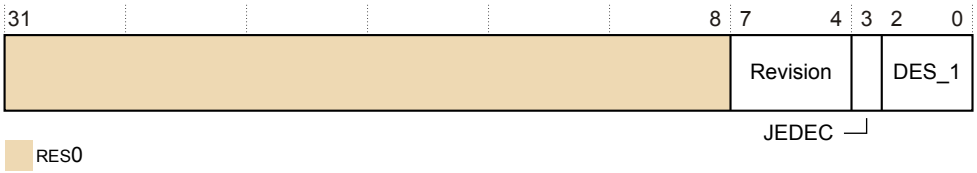


Figure D3-12 EDPIDR2 bit assignments

RES0, [31:8]

RES0 Reserved.

Revision, [7:4]

2 r1p0.

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`.

D3.14 EDPIDR3, External Debug Peripheral Identification Register 3

The EDPIDR3 provides information to identify an external debug component.

Bit field descriptions

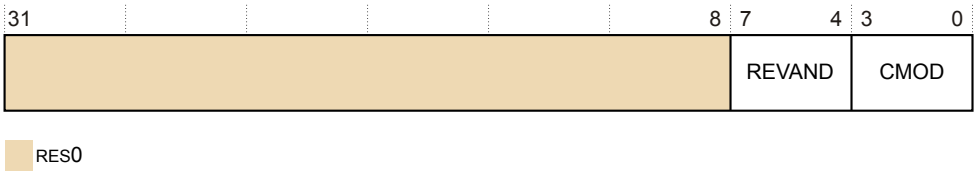


Figure D3-13 EDPIDR3 bit assignments

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`.

D3.15 EDPIDR4, External Debug Peripheral Identification Register 4

The EDPIDR4 provides information to identify an external debug component.

Bit field descriptions

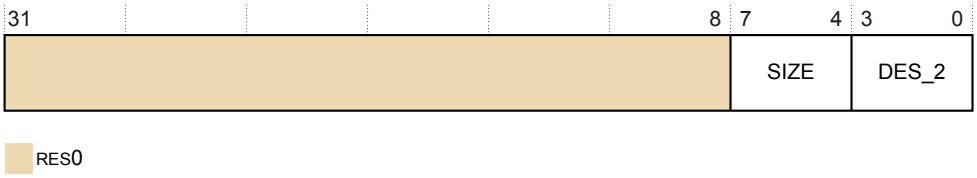


Figure D3-14 EDPIDR4 bit assignments

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`.

D3.16 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.

D3.17 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

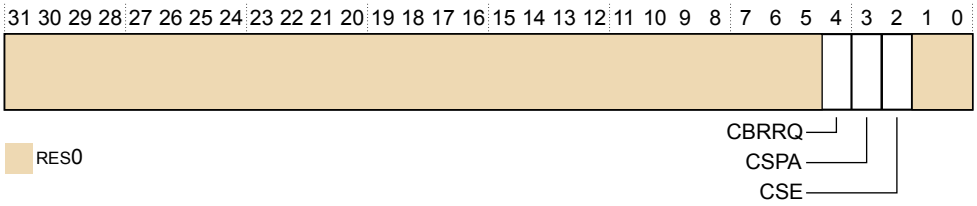


Figure D3-15 EDRCR bit assignments

RES0, [31:5]

RES0 Reserved.

CBRRQ, [4]

Allow imprecise entry to Debug state. The actions on writing to this bit are:

`0` No action.

`1` Allow imprecise entry to Debug state, for example by canceling pending bus accesses. Setting this bit to 1 allows a debugger to request imprecise entry to Debug state. An External Debug Request debug event must be pending before the debugger sets this bit to 1.

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.

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 EDRCR can be accessed through the external debug interface, offset `0x090`.

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-A55 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	`0x41453000`	*D4.4 PMCR, Performance Monitors Control Register* on page D4-620
c9	0	c12	1	PMCNTENSET	RW	32	UNK	Performance Monitors Count Enable Set Register
c9	0	c12	2	PMCNTENCLR	RW	32	UNK	Performance Monitors Count Enable Clear Register
c9	0	c12	3	PMOVSR	RW	32	UNK	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	If L2 is present: `0x67FFBFFF`. If L3 is present: `0x66FFBBFFF`. If L2 and L3 are not present: `0x663FBFFF`.	*D4.2 PMCEID0, Performance Monitors Common Event Identification Register 0* on page D4-614
c9	0	c12	7	PMCEID1	RO	32	If L2 and L3 are present: `0x00F2AE7F`. If L2 or L3 is present: `0x00F2A07F`. If L2 and L3 are not present: `0x00F2A07E`.	*D4.3 PMCEID1, Performance Monitors Common Event Identification Register 1* on page D4-618
c9	0	c14	4	PMCEID2	RO	32	UNK	Reserved
c9	0	c14	5	PMCEID3	RO	32	UNK	Reserved
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	UNK	Performance Monitors Interrupt Enable Set Register

Table D4-1 PMU register summary in the AArch32 Execution state (continued)

CRn	Op1	CRm	Op2	Name	Type	Width	Reset	Description
c9	0	c14	2	PMINTENCLR	RW	32	UNK	Performance Monitors Interrupt Enable Clear Register
c9	0	c14	3	PMOVSSET	RW	32	UNK	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	
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

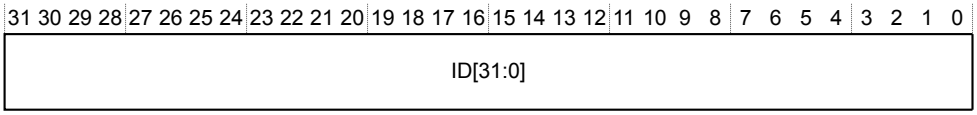


Figure D4-1 PMCEID0 bit assignments

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: 0 This event is not 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: 1 This event is implemented.
[25]	0x19	BUS_ACCESS	Bus access: 1 This event is implemented.

Table D4-2 PMU events (continued)

Bit	Event number	Event mnemonic	Description
[24]	0x18	L2D_CACHE_WB	L2 Data cache Write-Back: 0 This event is not implemented if the Cortex-A55 core has been configured without an L2 cache. 1 This event is implemented if the Cortex-A55 core has been configured with an L2 cache.
[23]	0x17	L2D_CACHE_REFILL	L2 Data cache refill: 0 This event is not implemented if the Cortex-A55 core has been configured without an L2 and L3 cache. If configured with only an L3 cache, the L3 event will become an L2 event. 1 This event is implemented if the Cortex-A55 core has been configured with an L2 or L3 cache.
[22]	0x16	L2D_CACHE	L2 Data cache access: 0 This event is not implemented if the Cortex-A55 core has been configured without an L2 and L3 cache. If configured with only an L3 cache, the L3 event will become an L2 event. 1 This event is implemented if the Cortex-A55 core has been configured with an L2 or L3 cache.
[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: 1 This event is implemented.
[14]	0x0E	BR_RETURN_RETIRED	Instruction architecturally executed, condition check pass - procedure return: 1 This event is implemented.

Table D4-2 PMU events (continued)

Bit	Event number	Event mnemonic	Description
[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.
[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: 1 This event is implemented.
[6]	0x06	LD_RETIRED	Instruction architecturally executed, condition check pass - load: 1 This event is 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

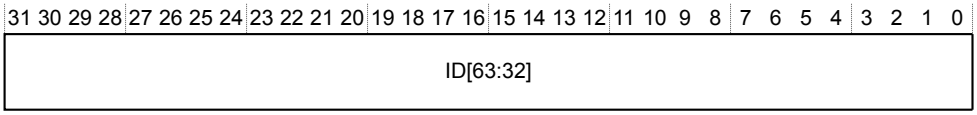


Figure D4-2 PMCEID1 bit assignments

ID[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 D4-3 PMU common events

Bit	Event number	Event mnemonic	Description
[23]	0x37	LL_CACHE_MISS_RD	Last Level cache miss, 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	Access to instruction TLB that caused a page table walk. 1 This event is implemented.
[20]	0x34	DTLB_WALK	Access to data TLB that caused a page table walk. 1 This event is implemented.
[17]	0x31	REMOTE_ACCESS	Access to another socket in a multi-socket system. 1 This event is implemented.
[16]	0x30	L2I_TLB	Attributable Level 2 instruction TLB access. 0 This event is not implemented.
[15]	0x2F	L2D_TLB	Attributable Level 2 data or unified TLB access. 1 This event is implemented.
[14]	0x2E	L2I_TLB_REFILL	Attributable Level 2 instruction TLB refill. 0 This event is not implemented.
[13]	0x2D	L2D_TLB_REFILL	Attributable Level 2 data or unified TLB refill. 1 This event is implemented.

Table D4-3 PMU common events (continued)

Bit	Event number	Event mnemonic	Description
[12]	0x2C	L3D_CACHE_WB	Attributable Level 3 data or unified cache write-back. 0 This event is not implemented.
[11]	0x2B	L3D_CACHE	Attributable Level 3 data or unified cache access. 1 This event is implemented if L2 and L3 are present. 0 This event is not implemented if L2 and L3 are not present.
[10]	0x2A	L3D_CACHE_REFILL	Attributable Level 3 data or unified cache refill. 1 This event is implemented if L2 and L3 are present. 0 This event is not implemented if L2 and L3 are not present.
[9]	0x29	L3D_CACHE_ALLOCATE	Attributable Level 3 data or unified cache allocation without refill. 1 This event is implemented if L2 and L3 are present. 0 This event is not implemented if L2 and L3 are not present.
[8]	0x28	L2I_CACHE_REFILL	Attributable Level 2 instruction cache refill. 0 This event is not implemented.
[7]	0x27	L2I_CACHE	Attributable Level 2 instruction cache access. 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 the frontend. 1 This event is implemented.
[2]	0x22	BR_MIS_PRED_RETIRED	Instruction architecturally executed, mispredicted branch. 1 This event is 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 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.

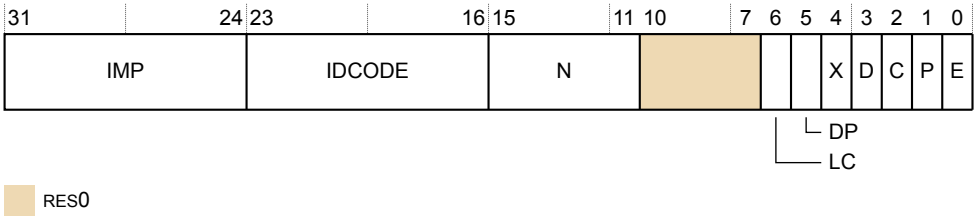


Figure D4-3 PMCR bit assignments

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:

`0x45` Cortex-A55 core.

N, [15:11]
Identifies the number of event counters implemented.

`0b0011` The core implements six event counters.
`0`

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.

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.

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:

`0` 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 D5-632.

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.

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 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-A55 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	`0x41453000`	*D5.4 PMCR_EL0, Performance Monitors Control Register, EL0* on page D5-632
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	UNK	*D5.2 PMCEID0_EL0, Performance Monitors Common Event Identification Register 0, EL0* on page D5-626
PMCEID1_EL0	RO	64	UNK	*D5.3 PMCEID1_EL0, Performance Monitors Common Event Identification Register 1, EL0* on page D5-630
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

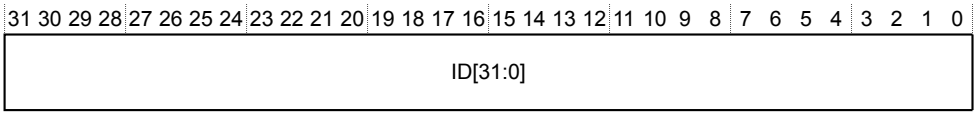


Figure D5-1 PMCEID0_EL0 bit assignments

ID[31:0], [31:0]

Common architectural and microarchitectural feature events that can be counted by the PMU event counters.

The following table shows the PMCEID0_EL0 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 D5-2 PMU events

Bit	Event number	Event mnemonic	Description
[31]	0x1F	L1D_CACHE_ALLOCATE	L1 Data cache allocate: 0 This event is not 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: 1 This event is implemented.
[25]	0x19	BUS_ACCESS	Bus access: 1 This event is implemented.

Table D5-2 PMU events (continued)

Bit	Event number	Event mnemonic	Description
[24]	0x18	L2D_CACHE_WB	L2 Data cache Write-Back: 0 This event is not implemented if the Cortex-A55 core has been configured without an L2 cache. 1 This event is implemented if the Cortex-A55 core has been configured with an L2 cache.
[23]	0x17	L2D_CACHE_REFILL	L2 Data cache refill: 0 This event is not implemented if the Cortex-A55 core has been configured without an L2 and L3 cache. If configured with only an L3 cache, the L3 event will become an L2 event. 1 This event is implemented if the Cortex-A55 core has been configured with an L2 or L3 cache.
[22]	0x16	L2D_CACHE	L2 Data cache access: 0 This event is not implemented if the Cortex-A55 core has been configured without an L2 and L3 cache. If configured with only an L3 cache, the L3 event will become an L2 event. 1 This event is implemented if the Cortex-A55 core has been configured with an L2 or L3 cache.
[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: 1 This event is implemented.
[14]	0x0E	BR_RETURN_RETIRED	Instruction architecturally executed, condition check pass - procedure return: 1 This event is implemented.

Table D5-2 PMU events (continued)

Bit	Event number	Event mnemonic	Description
[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.
[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: 1 This event is implemented.
[6]	0x06	LD_RETIRED	Instruction architecturally executed, condition check pass - load: 1 This event is 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*.

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

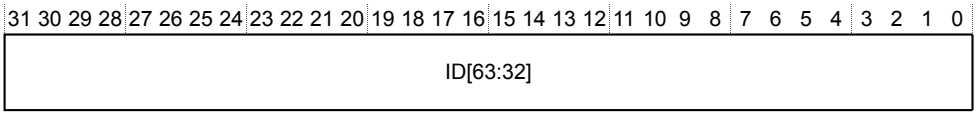


Figure D5-2 PMCEID1_EL0 bit assignments

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
[23]	0x37	LL_CACHE_MISS_RD	Last Level cache miss, 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	Access to instruction TLB that caused a page table walk. 1 This event is implemented.
[20]	0x34	DTLB_WALK	Access to data TLB that caused a page table walk. 1 This event is implemented.
[17]	0x31	REMOTE_ACCESS	Access to another socket in a multi-socket system. 1 This event is implemented.
[16]	0x30	L2I_TLB	Attributable Level 2 instruction TLB access. 0 This event is not implemented.
[15]	0x2F	L2D_TLB	Attributable Level 2 data or unified TLB access. 1 This event is implemented.
[14]	0x2E	L2I_TLB_REFILL	Attributable Level 2 instruction TLB refill. 0 This event is not implemented.
[13]	0x2D	L2D_TLB_REFILL	Attributable Level 2 data or unified TLB refill. 1 This event is implemented.

Table D5-3 PMU common events (continued)

Bit	Event number	Event mnemonic	Description
[12]	0x2C	L3D_CACHE_WB	Attributable Level 3 data or unified cache write-back. 0 This event is not implemented.
[11]	0x2B	L3D_CACHE	Attributable Level 3 data or unified cache access. 1 This event is implemented if L2 and L3 are present. 0 This event is not implemented if L2 and L3 are not present.
[10]	0x2A	L3D_CACHE_REFILL	Attributable Level 3 data or unified cache refill. 1 This event is implemented if L2 and L3 are present. 0 This event is not implemented if L2 and L3 are not present.
[9]	0x29	L3D_CACHE_ALLOCATE	Attributable Level 3 data or unified cache allocation without refill. 1 This event is implemented if L2 and L3 are present. 0 This event is not implemented if L2 and L3 are not present.
[8]	0x28	L2I_CACHE_REFILL	Attributable Level 2 instruction cache refill. 0 This event is not implemented.
[7]	0x27	L2I_CACHE	Attributable Level 2 instruction cache access. 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 the frontend. 1 This event is implemented.
[2]	0x22	BR_MIS_PRED_RETIRED	Instruction architecturally executed, mispredicted branch. 1 This event is 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*.

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

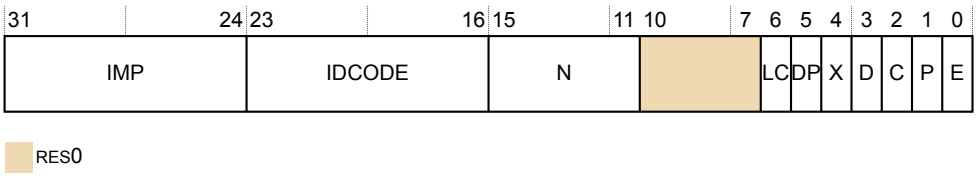


Figure D5-3 PMCR_EL0 bit assignments

IMP, [31:24]
Implementer code:

`0x41` ARM.

This is a read-only field.

IDCODE, [23:16]
Identification code:

`0x45` Cortex-A55.

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	*D7.2 PMPCSSR, Snapshot Program Counter Sample Register* on page D7-653
0x604	PMPCSSR_HI	RO	
0x608	PMCIDSSR	RO	*D7.3 PMCIDSSR, Snapshot CONTEXTIDR_EL1 Sample Register* on page D7-654
0x60C	PMCID2SSR	RO	*D7.4 PMCID2SSR, Snapshot CONTEXTIDR_EL2 Sample Register* on page D7-655
0x610	PMSSSR	RO	*D7.5 PMSSSR, PMU Snapshot Status Register* on page D7-656
0x614	PMOVSSR	RO	*D7.6 PMOVSSR, PMU Overflow Status Snapshot Register* on page D7-657
0x618	PMCCNTSR_LO	RO	*D7.7 PMCCNTSR, PMU Cycle Counter Snapshot Register* on page D7-658
0x61C	PMCCNTSR_HI	RO	
0x620+ 4×n	PMEVCNTSR<n>	RO	*D7.8 PMEVCNTSR 0-5, PMU Cycle Counter Snapshot Registers* on page D7-659
0x6F0	PMSSCR	WO	*D7.9 PMSSCR, PMU Snapshot Capture Register* on page D7-660
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 D6-640
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 D5-626
0xE24	PMCEID1	RO	*D5.3 PMCEID1_EL0, Performance Monitors Common Event Identification Register 1, EL0* on page D5-630
0xE28	PMCEID2	RO	Reserved
0xE2C	PMCEID3	RO	Reserved
0xFA4	-	-	Reserved
0xFA8	PMDEVAFF0	RO	*B2.83 MPIDR_EL1, Multiprocessor Affinity Register, EL1* on page B2-404
0xFAC	PMDEVAFF1	RO	*B2.83 MPIDR_EL1, Multiprocessor Affinity Register, EL1* on page B2-404
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 D6-649
0xFD4	PMPIDR5	RO	*D6.12 PMPIDRn, Performance Monitors Peripheral Identification Register 5-7* on page D6-650
0xFD8	PMPIDR6	RO	
0xFDC	PMPIDR7	RO	
0xFE0	PMPIDR0	RO	*D6.7 PMPIDR0, Performance Monitors Peripheral Identification Register 0* on page D6-645
0xFE4	PMPIDR1	RO	*D6.8 PMPIDR1, Performance Monitors Peripheral Identification Register 1* on page D6-646
0xFE8	PMPIDR2	RO	*D6.9 PMPIDR2, Performance Monitors Peripheral Identification Register 2* on page D6-647

Table D6-1 Memory-mapped PMU register summary (continued)

Offset	Name	Type	Description
0xFEC	PMPIDR3	RO	*D6.10 PMPIDR3, Performance Monitors Peripheral Identification Register 3* on page D6-648
0xFF0	PMCIDR0	RO	*D6.3 PMCIDR0, Performance Monitors Component Identification Register 0* on page D6-641
0xFF4	PMCIDR1	RO	*D6.4 PMCIDR1, Performance Monitors Component Identification Register 1* on page D6-642
0xFF8	PMCIDR2	RO	*D6.5 PMCIDR2, Performance Monitors Component Identification Register 2* on page D6-643
0xFFC	PMCIDR3	RO	*D6.6 PMCIDR3, Performance Monitors Component Identification Register 3* on page D6-644

D6.2 PMCFGR, Performance Monitors Configuration Register

The PMCFGR contains PMU specific configuration data.

Bit field descriptions

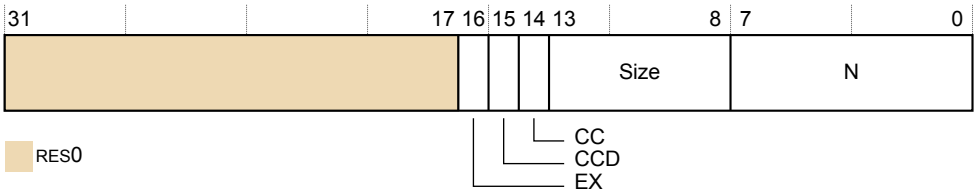


Figure D6-1 PMCFGR bit assignments

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

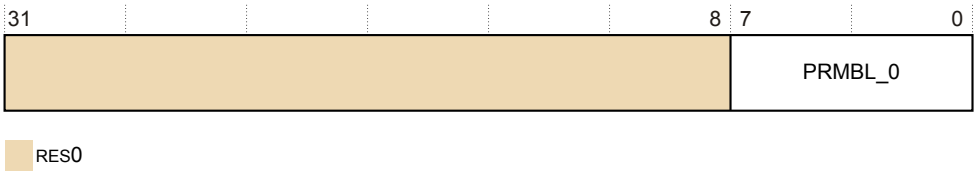


Figure D6-2 PMCIDR0 bit assignments

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

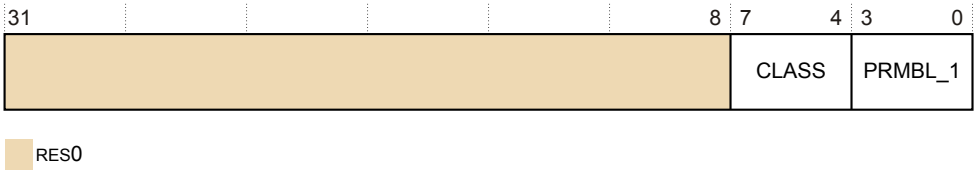


Figure D6-3 PMCIDR1 bit assignments

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

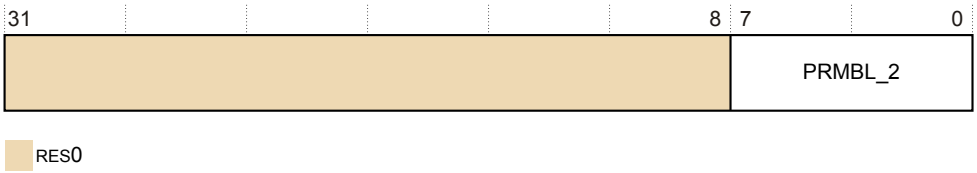


Figure D6-4 PMCIDR2 bit assignments

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

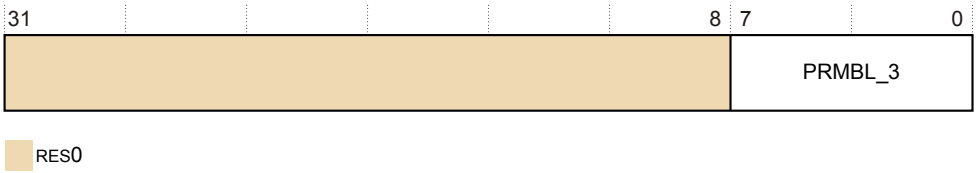


Figure D6-5 PMCIDR3 bit assignments

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

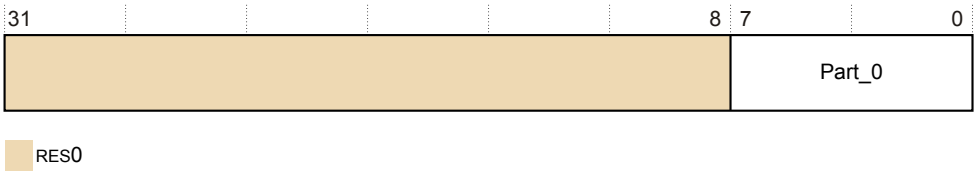


Figure D6-6 PMPIDR0 bit assignments

RES0, [31:8]

RES0 Reserved.

Part_0, [7:0]

`0x05` 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

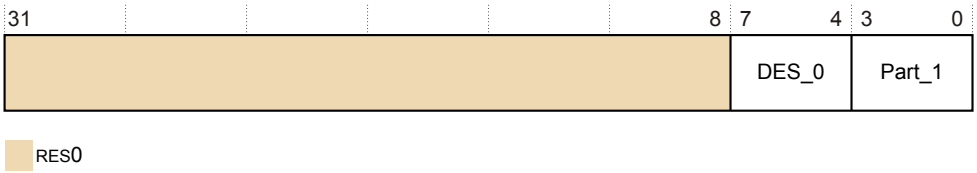


Figure D6-7 PMPIDR1 bit assignments

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

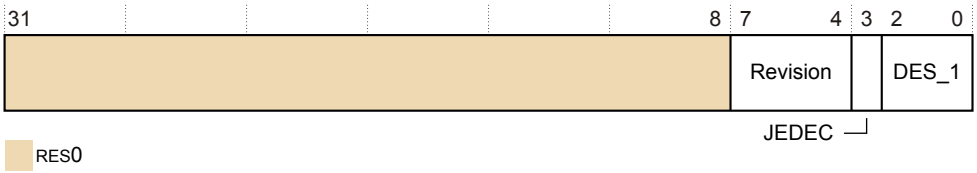


Figure D6-8 PMPIDR2 bit assignments

RES0, [31:8]

RES0 Reserved.

Revision, [7:4]

`0x2` r1p0.

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

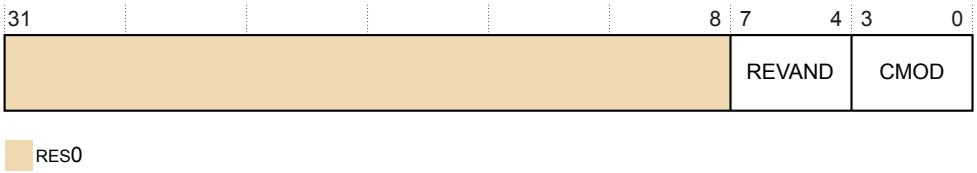


Figure D6-9 PMPIDR3 bit assignments

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

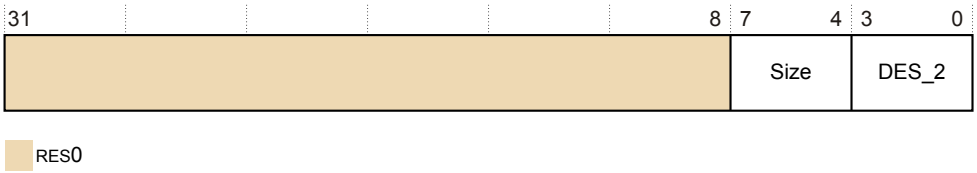


Figure D6-10 PMPIDR4 bit assignments

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
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:

D7.1 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 D7-1 PMU snapshot register summary

Offset	Name	Type	Width	Description
0x600	PMPCSSR_LO	RO	32	*D7.2 PMPCSSR, Snapshot Program Counter Sample Register* on page D7-653
0x604	PMPCSSR_HI	RO	32	
0x608	PMPCIDSSR	RO	32	*D7.3 PMCIDSSR, Snapshot CONTEXTIDR_EL1 Sample Register* on page D7-654
0x60C	PMPCID2SSR	RO	32	*D7.4 PMCID2SSR, Snapshot CONTEXTIDR_EL2 Sample Register* on page D7-655
0x610	PMSSSR	RO	32	*D7.5 PMSSSR, PMU Snapshot Status Register* on page D7-656
0x614	PMOVSSR	RO	32	*D7.6 PMOVSSR, PMU Overflow Status Snapshot Register* on page D7-657
0x618	PMCCNTSR_LO	RO	32	*D7.7 PMCCNTSR, PMU Cycle Counter Snapshot Register* on page D7-658
0x61C	PMCCNTSR_HI	RO	32	
0x620 + 4×n	PMEVCNTSR<n>	RO	32	*D7.8 PMEVCNTSR 0-5, PMU Cycle Counter Snapshot Registers* on page D7-659
0x6F0	PMSSCR	WO	32	*D7.9 PMSSCR, PMU Snapshot Capture Register* on page D7-660

D7.2 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.

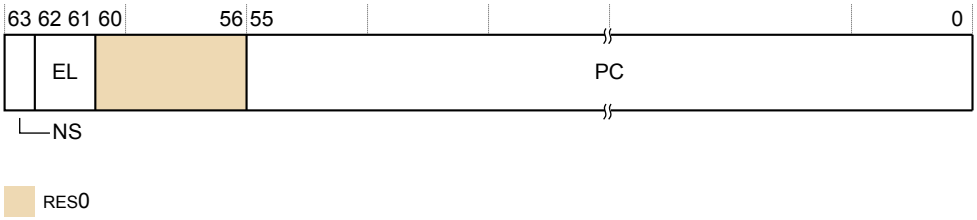


Figure D7-1 PMPCSSR bit assignments

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*.

D7.3 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*.

D7.4 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*.

D7.5 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.

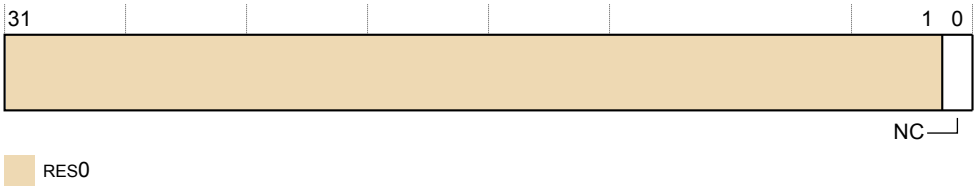


Figure D7-2 PMSSSR bit assignments

[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*.

D7.6 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*.

D7.7 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*.

D7.8 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*.

D7.9 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.

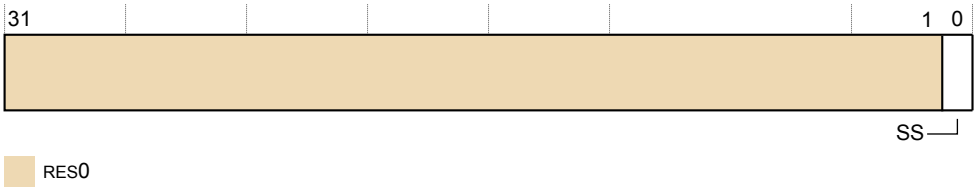


Figure D7-3 PMSSCR bit assignments

[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 D8
ETM registers

This chapter describes the ETM registers.

It contains the following sections:

D8.1 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 D8-1 ETM trace unit register summary

Offset	Name	Type	Reset	Description
0x004	TRCPRGCTLR	RW	0x00000000	*D8.60 TRCPRGCTLR, Programming Control Register* on page D8-736
0x00C	TRCSTATR	RO	0x00000003	*D8.68 TRCSTATR, Status Register* on page D8-745
0x010	TRCCONFIGR	RW	UNK	*D8.20 TRCCONFIGR, Trace Configuration Register* on page D8-689
0x018	TRCAUXCTLR	RW	0x00000000	*D8.5 TRCAUXCTLR, Auxiliary Control Register* on page D8-671
0x020	TRCEVENTCTL0R	RW	UNK	*D8.26 TRCEVENTCTL0R, Event Control 0 Register* on page D8-697
0x024	TRCEVENTCTL1R	RW	UNK	*D8.27 TRCEVENTCL1R, Event Control 1 Register* on page D8-699
0x02C	TRCSTALLCTLR	RW	UNK	*D8.67 TRCSTALLCTLR, Stall Control Register* on page D8-744
0x030	TRCTSCTLR	RW	UNK	*D8.71 TRCTSCTLR, Global Timestamp Control Register* on page D8-748
0x034	TRCSYNCPR	RW	UNK	*D8.69 TRCSYNCPR, Synchronization Period Register* on page D8-746
0x038	TRCCCCTLR	RW	UNK	*D8.7 TRCCCCTLR, Cycle Count Control Register* on page D8-674
0x03C	TRCBBCTLR	RW	UNK	*D8.6 TRCBBCTLR, Branch Broadcast Control Register* on page D8-673
0x040	TRCTRACEIDR	RW	UNK	*D8.70 TRCTRACEIDR, Trace ID Register* on page D8-747
0x080	TRCVICTLR	RW	UNK	*D8.72 TRCVICTLR, ViewInst Main Control Register* on page D8-749
0x084	TRCVIIECTLR	RW	UNK	*D8.73 TRCVIIECTLR, ViewInst Include-Exclude Control Register* on page D8-751
0x088	TRCVISSCTLR	RW	UNK	*D8.74 TRCVISSCTLR, ViewInst Start-Stop Control Register* on page D8-752
0x100	TRCSEQEVR0	RW	UNK	*D8.62 TRCSEQEVRn, Sequencer State Transition Control Registers 0-2* on page D8-738
0x104	TRCSEQEVR1	RW	UNK	*D8.62 TRCSEQEVRn, Sequencer State Transition Control Registers 0-2* on page D8-738
0x108	TRCSEQEVR2	RW	UNK	*D8.62 TRCSEQEVRn, Sequencer State Transition Control Registers 0-2* on page D8-738
0x118	TRCSEQRSTEVR	RW	UNK	*D8.63 TRCSEQRSTEVR, Sequencer Reset Control Register* on page D8-740
0x11C	TRCSEQSTR	RW	UNK	*D8.64 TRCSEQSTR, Sequencer State Register* on page D8-741
0x120	TRCEXTINSELR	RW	UNK	*D8.28 TRCEXTINSELR, External Input Select Register* on page D8-700
0x140	TRCCNTRLDVR0	RW	UNK	*D8.18 TRCCNTRLDVRn, Counter Reload Value Registers 0-1* on page D8-687

Table D8-1 ETM trace unit register summary (continued)

Offset	Name	Type	Reset	Description
0x144	TRCCNTRLDVR1	RW	UNK	*D8.18 TRCCNTRLDVRn, Counter Reload Value Registers 0-1* on page D8-687
0x150	TRCCNTCTLR0	RW	UNK	*D8.16 TRCCNTCTLR0, Counter Control Register 0* on page D8-683
0x154	TRCCNTCTLR1	RW	UNK	*D8.17 TRCCNTCTLR1, Counter Control Register 1* on page D8-685
0x160	TRCCNTVR0	RW	UNK	*D8.19 TRCCNTVRn, Counter Value Registers 0-1* on page D8-688
0x164	TRCCNTVR1	RW	UNK	*D8.19 TRCCNTVRn, Counter Value Registers 0-1* on page D8-688
0x180	TRCIDR8	RO	0x00000000	*D8.35 TRCIDR8, ID Register 8* on page D8-711
0x184	TRCIDR9	RO	0x00000000	*D8.36 TRCIDR9, ID Register 9* on page D8-712
0x188	TRCIDR10	RO	0x00000000	*D8.37 TRCIDR10, ID Register 10* on page D8-713
0x18C	TRCIDR11	RO	0x00000000	*D8.38 TRCIDR11, ID Register 11* on page D8-714
0x190	TRCIDR12	RO	0x00000000	*D8.39 TRCIDR12, ID Register 12* on page D8-715
0x194	TRCIDR13	RO	0x00000000	*D8.40 TRCIDR13, ID Register 13* on page D8-716
0x1C0	TRCIMSPEC0	RW	0x00000000	*D8.41 TRCIMSPEC0, Implementation Specific Register 0* on page D8-717
0x1E0	TRCIDR0	RO	0x28000EA1	*D8.29 TRCIDR0, ID Register 0* on page D8-701
0x1E4	TRCIDR1	RO	0x41001422	*D8.30 TRCIDR1, ID Register 1* on page D8-703
0x1E8	TRCIDR2	RO	0x20001048	*D8.31 TRCIDR2, ID Register 2* on page D8-704
0x1EC	TRCIDR3	RO	0x0D7B0004	*D8.32 TRCIDR3, ID Register 3* on page D8-706
0x1F0	TRCIDR4	RO	0x11170004	*D8.33 TRCIDR4, ID Register 4* on page D8-708
0x1F4	TRCIDR5	RO	0x2883842F	*D8.34 TRCIDR5, ID Register 5* on page D8-709
0x200	TRCRSCTLRn	RW	UNK	*D8.61 TRCRSCTLRn, Resource Selection Control Registers 2-16* on page D8-737, n is 2, 15
0x280	TRCSSCCR0	RW	UNK	*D8.65 TRCSSCCR0, Single-Shot Comparator Control Register 0* on page D8-742
0x2A0	TRCSSCSR0	RW	UNK	*D8.66 TRCSSCSR0, Single-Shot Comparator Status Register 0* on page D8-743
0x300	TRCOSLAR	WO	0x00000001	*D8.50 TRCOSLAR, OS Lock Access Register* on page D8-726
0x304	TRCOSLSR	RO	0x0000000A	*D8.51 TRCOSLSR, OS Lock Status Register* on page D8-727
0x310	TRCPDCR	RW	0x00000000	*D8.52 TRCPDCR, Power Down Control Register* on page D8-728
0x314	TRCPDSR	RO	0x00000013	*D8.53 TRCPDSR, Power Down Status Register* on page D8-729
0x400	TRCACVRn	RW	UNK	*D8.3 TRCACVRn, Address Comparator Value Registers 0-7* on page D8-669
0x480	TRCACATRn	RW	UNK	*D8.2 TRCACATRn, Address Comparator Access Type Registers 0-7* on page D8-667
0x600	TRCCIDCVR0	RW	UNK	*D8.9 TRCCIDCVR0, Context ID Comparator Value Register 0* on page D8-676

Table D8-1 ETM trace unit register summary (continued)

Offset	Name	Type	Reset	Description
0x640	TRCVMIDCVR0	RW	UNK	*D8.75 TRCVMIDCVR0, VMID Comparator Value Register 0* on page D8-753
0x680	TRCCIDCCTLR0	RW	UNK	*D8.8 TRCCIDCCTLR0, Context ID Comparator Control Register 0* on page D8-675
0x688	TRCVMIDCCTRL0	RW	UNK	Virtual context identifier Comparator Control Register 0
0xEE4	TRCITATBIDR	RW	UNK	*D8.42 TRCITATBIDR, Integration ATB Identification Register* on page D8-718
0xEEC	TRCITIDATAR	WO	UNK	*D8.46 TRCITIDATAR, Integration Instruction ATB Data Register* on page D8-722
0xEF4	TRCITIATBINR	RO	UNK	*D8.44 TRCITIATBINR, Integration Instruction ATB In Register* on page D8-720
0xEFC	TRCITIATBOUTR	WO	UNK	*D8.45 TRCITIATBOUTR, Integration Instruction ATB Out Register* on page D8-721
0xF00	TRCITCTRL	RW	0x00000000	*D8.43 TRCITCTRL, Integration Mode Control Register* on page D8-719
0xFA0	TRCCLAIMSET	RW	UNK	*D8.15 TRCCLAIMSET, Claim Tag Set Register* on page D8-682
0xFA4	TRCCLAIMCLR	RW	0x00000000	*D8.14 TRCCLAIMCLR, Claim Tag Clear Register* on page D8-681
0xFA8	TRCDEVAFF0	RO	UNK	*D8.21 TRCDEVAFF0, Device Affinity Register 0* on page D8-691
0xFAC	TRCDEVAFF1	RO	UNK	*D8.22 TRCDEVAFF1, Device Affinity Register 1* on page D8-693
0xFB0	TRCLAR	WO	UNK	*D8.47 TRCLAR, Software Lock Access Register* on page D8-723
0xFB4	TRCLSR	RO	0x00000000	*D8.48 TRCLSR, Software Lock Status Register* on page D8-724
0xFB8	TRCAUTHSTATUS	RO	UNK	*D8.4 TRCAUTHSTATUS, Authentication Status Register* on page D8-670
0xFBC	TRCDEVARCH	RO	0x47724A13	*D8.23 TRCDEVARCH, Device Architecture Register* on page D8-694
0xFC8	TRCDEVID	RO	0x00000000	*D8.24 TRCDEVID, Device ID Register* on page D8-695
0xFCC	TRCDEVTYPE	RO	0x00000013	*D8.25 TRCDEVTYPE, Device Type Register* on page D8-696
0xFE0	TRCPIDR0	RO	0x0000000A	*D8.54 TRCPIDR0, ETM Peripheral Identification Register 0* on page D8-730
0xFE4	TRCPIDR1	RO	0x000000BD	*D8.55 TRCPIDR1, ETM Peripheral Identification Register 1* on page D8-731
0xFE8	TRCPIDR2	RO	0x0000002B	*D8.56 TRCPIDR2, ETM Peripheral Identification Register 2* on page D8-732
0xFEC	TRCPIDR3	RO	0x00000000	*D8.57 TRCPIDR3, ETM Peripheral Identification Register 3* on page D8-733
0xFD0	TRCPIDR4	RO	0x00000004	*D8.58 TRCPIDR4, ETM Peripheral Identification Register 4* on page D8-734
0xFD4-0xFDC	TRCPIDRn	RO	0x00000000	*D8.59 TRCPIDRn, ETM Peripheral Identification Registers 5-7* on page D8-735
0xFF0	TRCCIDR0	RO	0x0000000D	*D8.10 TRCCIDR0, ETM Component Identification Register 0* on page D8-677

Table D8-1 ETM trace unit register summary (continued)

Offset	Name	Type	Reset	Description
0xFF4	TRCCIDR1	RO	0x00000090	*D8.11 TRCCIDR1, ETM Component Identification Register 1* on page D8-678
0xFF8	TRCCIDR2	RO	0x00000005	*D8.12 TRCCIDR2, ETM Component Identification Register 2* on page D8-679
0xFFC	TRCCIDR3	RO	0x000000B1	*D8.13 TRCCIDR3, ETM Component Identification Register 3* on page D8-680

D8.2 TRCACATRn, Address Comparator Access Type Registers 0-7

The TRCACATRn control the access for the corresponding address comparators.

Bit field descriptions

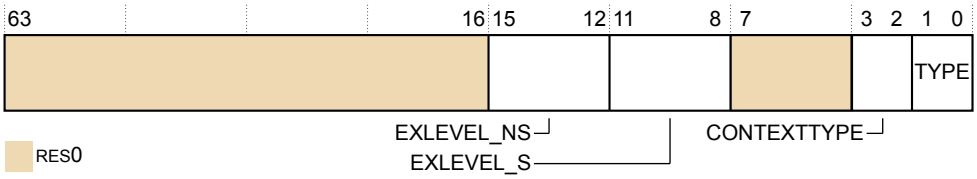


Figure D8-1 TRCACATRn bit assignments

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`.

D8.3 TRCACVRn, Address Comparator Value Registers 0-7

The TRCACVRn indicate the address for the address comparators.

Bit field descriptions

Figure D8-2 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`.

D8.4 TRCAUTHSTATUS, Authentication Status Register

The TRCAUTHSTATUS indicates the current level of tracing permitted by the system.

Bit field descriptions

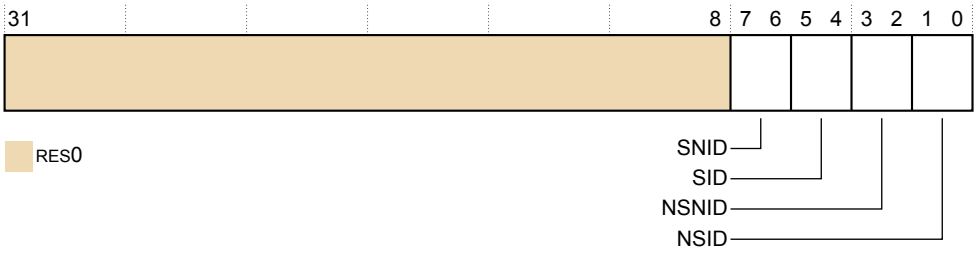


Figure D8-3 TRCAUTHSTATUS bit assignments

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`.

D8.5 TRCAUXCTLR, Auxiliary Control Register

The TRCAUXCTLR provides IMPLEMENTATION DEFINED configuration and control options.

Bit field descriptions

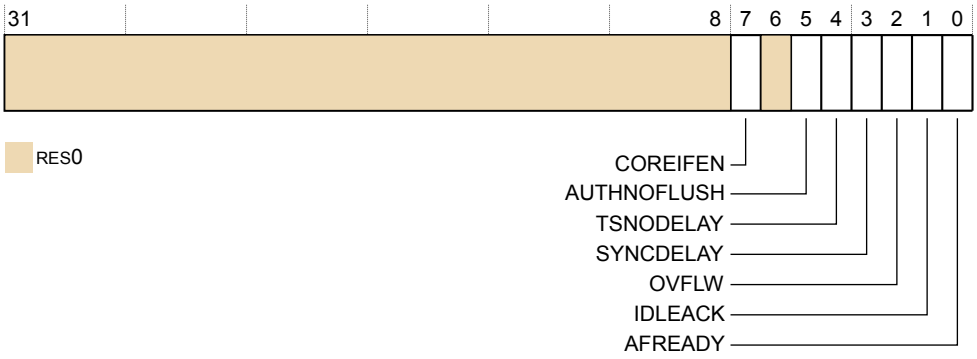


Figure D8-4 TRCAUXCTLR bit assignments

RES0, [31:8]

RES0 Reserved.

COREIFEN, [7]

Keep core interface enabled regardless of trace enable register state. The possible values are:

0 Core interface enabled is set by trace enable register state.

1 Enable core interface, regardless of trace enable register state.

RES0, [6]

RES0 Reserved.

AUTHNOFLUSH, [5]

Do not flush trace on de-assertion of authentication inputs. 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.

TSNODELAY, [4]

Do not delay timestamp insertion based on FIFO depth. 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.

SYNCDELAY, [3]

Delay periodic synchronization if FIFO is more than half-full. 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.

OVFLW, [2]

Force overflow if synchronization is not completed when second synchronization becomes due. 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.

IDLEACK, [1]

Force idle-drain acknowledge high, CPU does not wait for trace to drain before entering WFX state. 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.

AFREADY, [0]

Always respond to AFREADY immediately. Does not have any interaction with FIFO draining, even in WFI state. 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.

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 TRCAUXCTLR can be accessed through the external debug interface, offset `0x018`.

D8.6 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

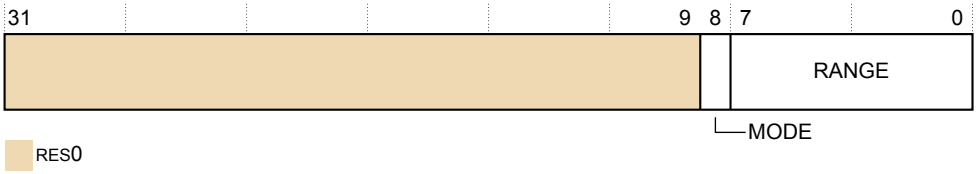


Figure D8-5 TRCBBCTLR bit assignments

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`.

D8.7 TRCCCCTLR, Cycle Count Control Register

The TRCCCCTLR sets the threshold value for cycle counting.

Bit field descriptions

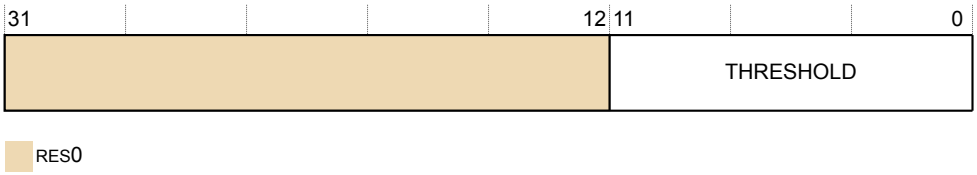


Figure D8-6 TRCCCCTLR bit assignments

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`.

D8.8 TRCCIDCCTLR0, Context ID Comparator Control Register 0

The TRCCIDCCTLR0 controls the mask value for the context ID comparators.

Bit field descriptions

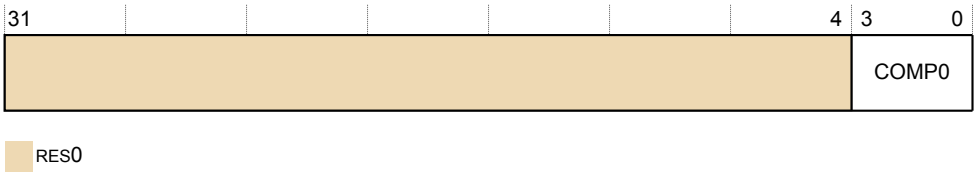


Figure D8-7 TRCCIDCCTLR0 bit assignments

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`.

D8.9 TRCCIDCVR0, Context ID Comparator Value Register 0

The TRCCIDCVR0 contains a Context ID value.

Bit field descriptions

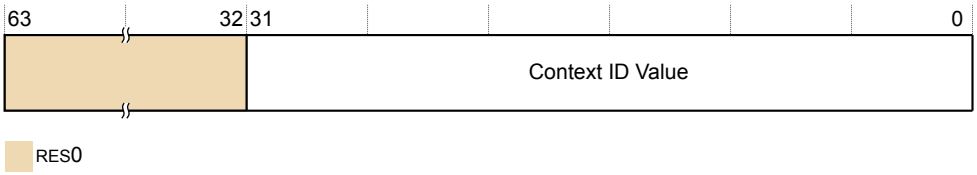


Figure D8-8 TRCCIDCVR0 bit assignments

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`.

D8.10 TRCCIDR0, ETM Component Identification Register 0

The TRCCIDR0 provides information to identify a trace component.

Bit field descriptions

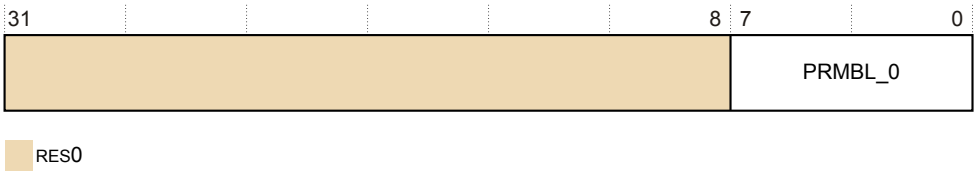


Figure D8-9 TRCCIDR0 bit assignments

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`.

D8.11 TRCCIDR1, ETM Component Identification Register 1

The TRCCIDR1 provides information to identify a trace component.

Bit field descriptions

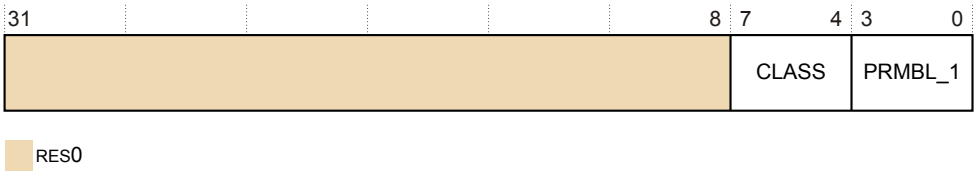


Figure D8-10 TRCCIDR1 bit assignments

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`.

D8.12 TRCCIDR2, ETM Component Identification Register 2

The TRCCIDR2 provides information to identify a CTI component.

Bit field descriptions

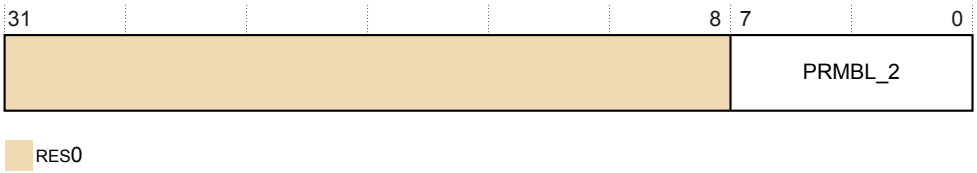


Figure D8-11 TRCCIDR2 bit assignments

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`.

D8.13 TRCCIDR3, ETM Component Identification Register 3

The TRCCIDR3 provides information to identify a trace component.

Bit field descriptions

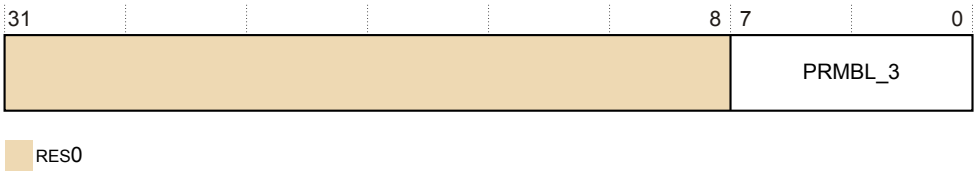


Figure D8-12 TRCCIDR3 bit assignments

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`.

D8.14 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

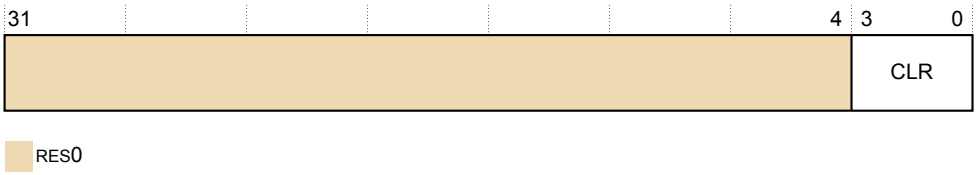


Figure D8-13 TRCCLAIMCLR bit assignments

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`.

D8.15 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

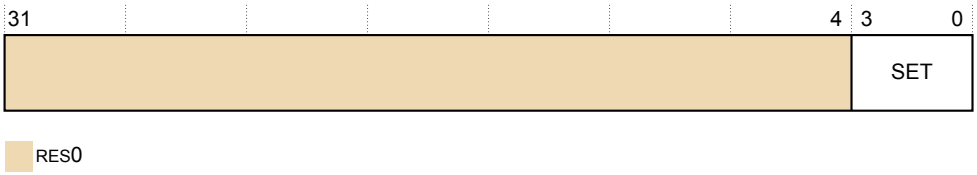


Figure D8-14 TRCCLAIMSET bit assignments

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`.

D8.16 TRCCNTCTLR0, Counter Control Register 0

The TRCCNTCTLR0 controls the counter.

Bit field descriptions

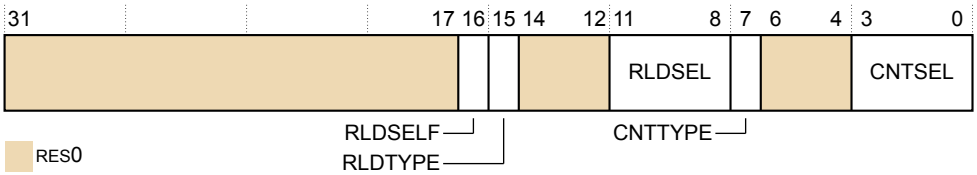


Figure D8-15 TRCCNTCTLR0 bit assignments

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`.

D8.17 TRCCNTCTLR1, Counter Control Register 1

The TRCCNTCTLR1 controls the counter.

Bit field descriptions

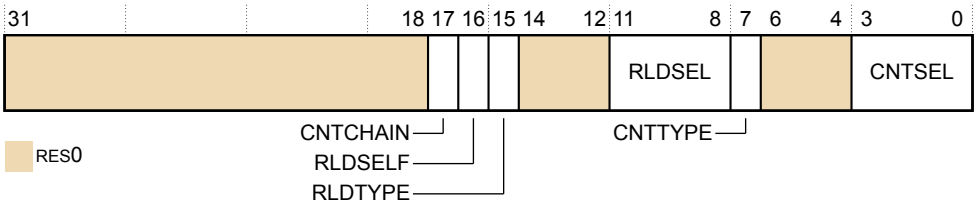


Figure D8-16 TRCCNTCTLR1 bit assignments

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`.

D8.18 TRCCNTRLDVRn, Counter Reload Value Registers 0-1

The TRCCNTRLDVRn define the reload value for the counter.

Bit field descriptions

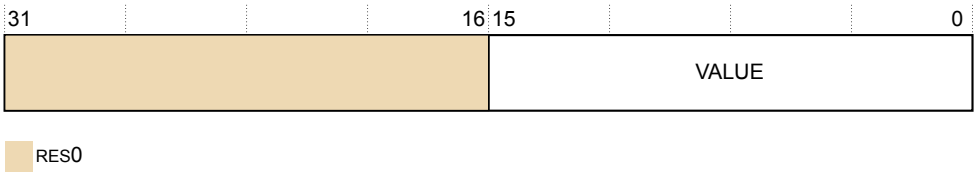


Figure D8-17 TRCCNTRLDVRn bit assignments

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`.

D8.19 TRCCNTVRn, Counter Value Registers 0-1

The TRCCNTRLDVRn contain the current counter value.

Bit field descriptions

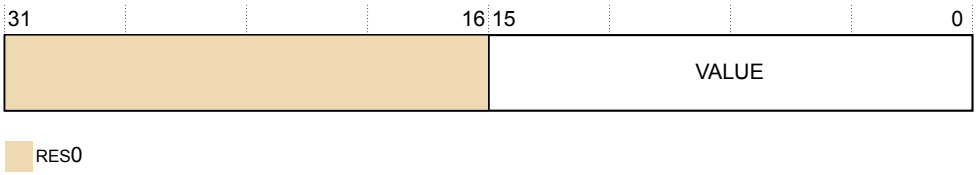


Figure D8-18 TRCCNTVRn bit assignments

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`.

D8.20 TRCCONFIGR, Trace Configuration Register

The TRCCONFIGR controls the tracing options.

Bit field descriptions

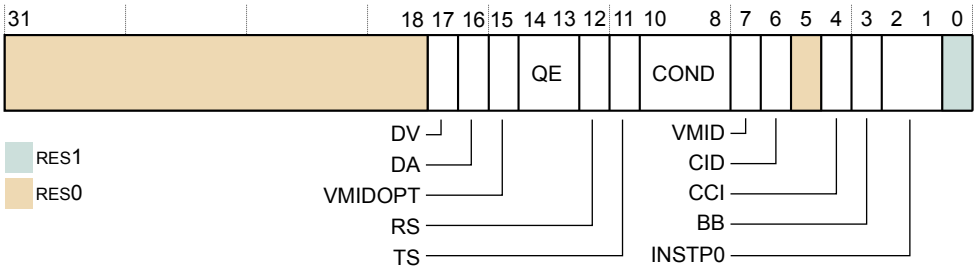


Figure D8-19 TRCCONFIGR bit assignments

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`.

D8.21 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

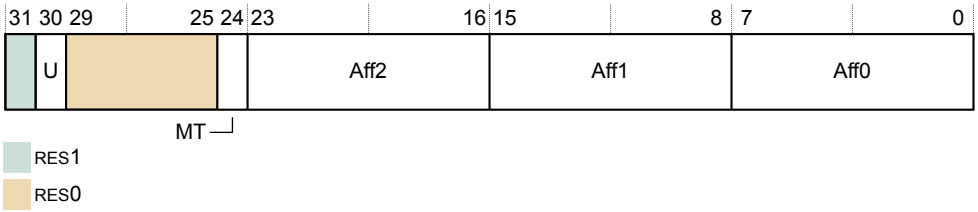


Figure D8-20 TRCDEVAFF0 bit assignments

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-A55 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`.

D8.22 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.

D8.23 TRCDEVARCH, Device Architecture Register

The TRCDEVARCH identifies the ETM trace unit as an ETMv4 component.

Bit field descriptions

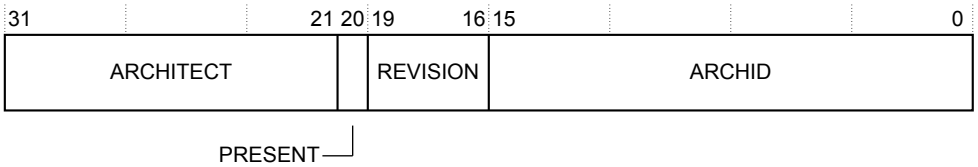


Figure D8-21 TRCDEVARCH bit assignments

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`.

D8.24 TRCDEVID, Device ID Register

The TRCDEVID indicates the capabilities of the ETM trace unit.

Bit field descriptions

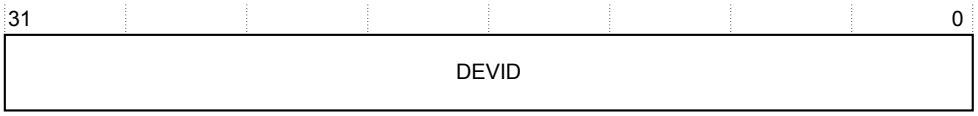


Figure D8-22 TRCDEVID bit assignments

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`.

D8.25 TRCDEVTYPE, Device Type Register

The TRCDEVTYPE indicates the type of the component.

Bit field descriptions

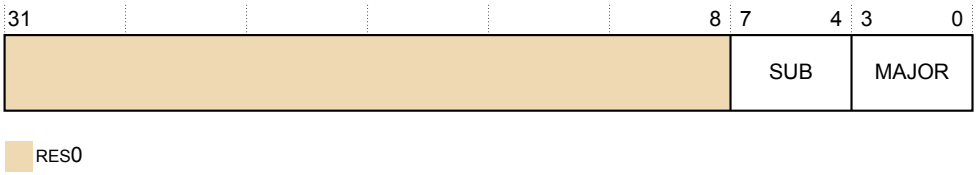


Figure D8-23 TRCDEVTYPE bit assignments

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`.

D8.26 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

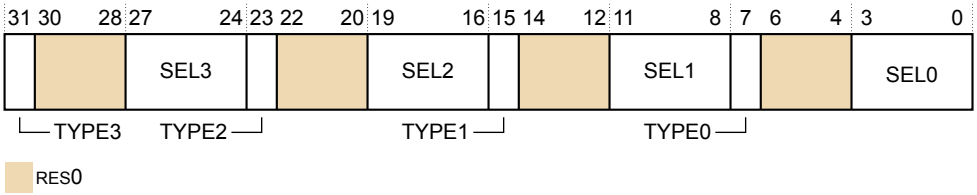


Figure D8-24 TRCEVENTCL0R bit assignments

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`.

D8.27 TRCEVENTCL1R, Event Control 1 Register

The TRCEVENTCTL1R controls the behavior of the events that TRCEVENTCTL0R selects.

Bit field descriptions

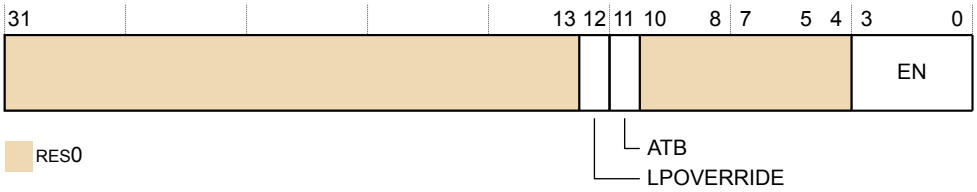


Figure D8-25 TRCEVENTCL1R bit assignments

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`.

D8.28 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

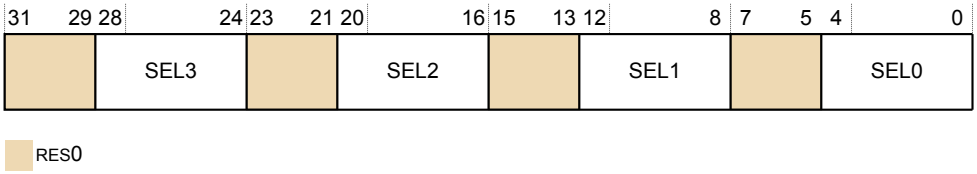


Figure D8-26 TRCEXTINSELR bit assignments

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`.

D8.29 TRCIDR0, ID Register 0

The TRCIDR0 returns the tracing capabilities of the ETM trace unit.

Bit field descriptions

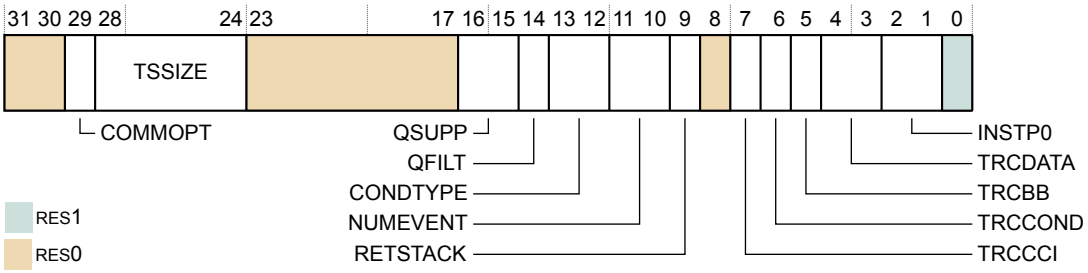


Figure D8-27 TRCIDR0 bit assignments

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`.

D8.30 TRCIDR1, ID Register 1

The TRCIDR1 returns the base architecture of the trace unit.

Bit field descriptions

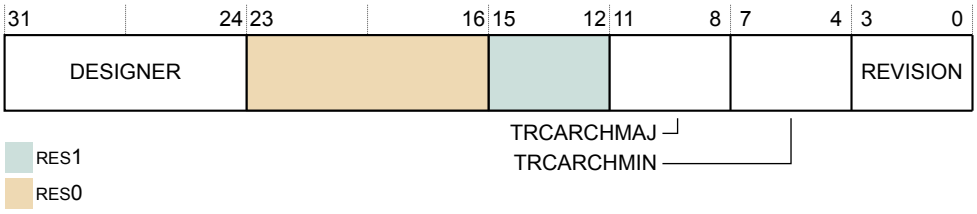


Figure D8-28 TRCIDR1 bit assignments

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:

2 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`.

D8.31 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

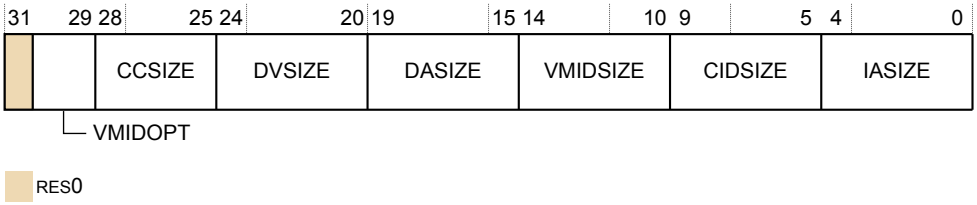


Figure D8-29 TRCIDR2 bit assignments

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`.

D8.32 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

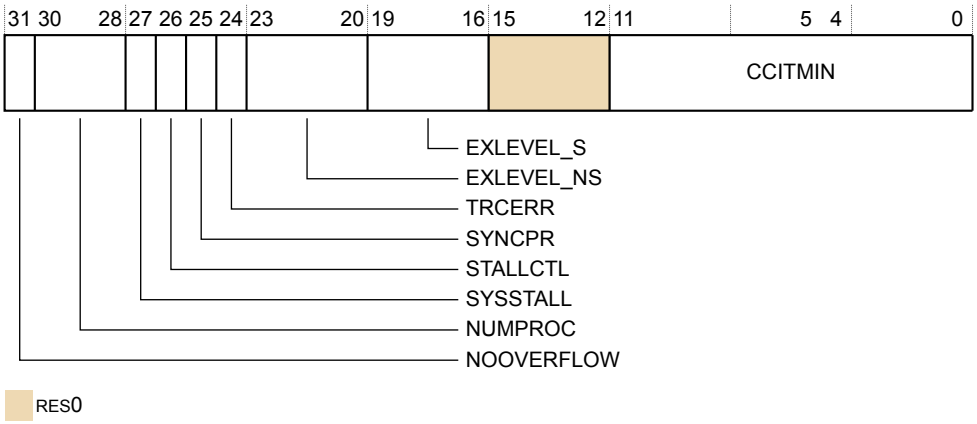


Figure D8-30 TRCIDR3 bit assignments

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`.

D8.33 TRCIDR4, ID Register 4

The TRCIDR4 indicates the resources available in the ETM trace unit.

Bit field descriptions

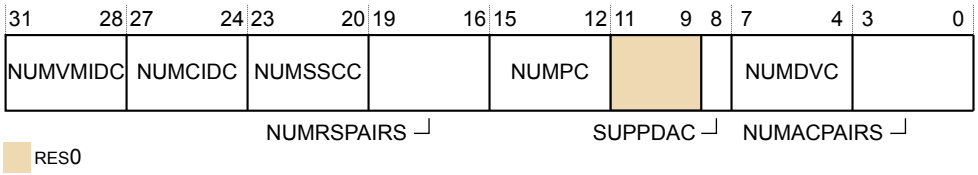


Figure D8-31 TRCIDR4 bit assignments

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`.

D8.34 TRCIDR5, ID Register 5

The TRCIDR5 returns how many resources the trace unit supports.

Bit field descriptions

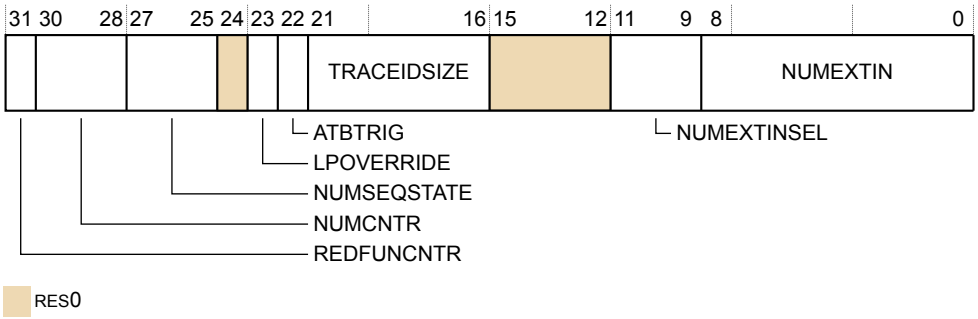


Figure D8-32 TRCIDR5 bit assignments

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:

`0x1E` 30 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`.

D8.35 TRCIDR8, ID Register 8

The TRCIDR8 returns the maximum speculation depth of the instruction trace stream.

Bit field descriptions

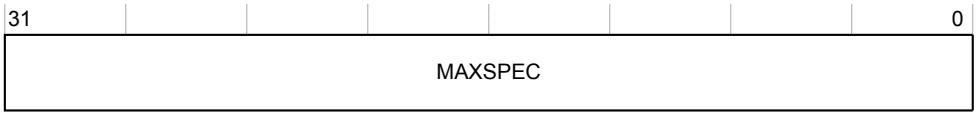


Figure D8-33 TRCIDR8 bit assignments

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`.

D8.36 TRCIDR9, ID Register 9

The TRCIDR9 returns the number of P0 right-hand keys that the trace unit can use.

Bit field descriptions

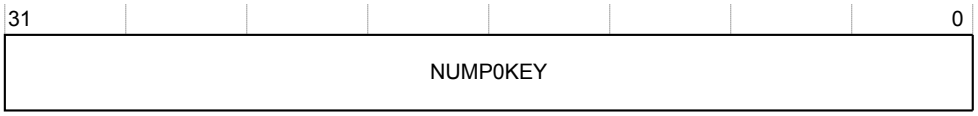


Figure D8-34 TRCID9 bit assignments

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`.

D8.37 TRCIDR10, ID Register 10

The TRCIDR10 returns the number of P1 right-hand keys that the trace unit can use.

Bit field descriptions

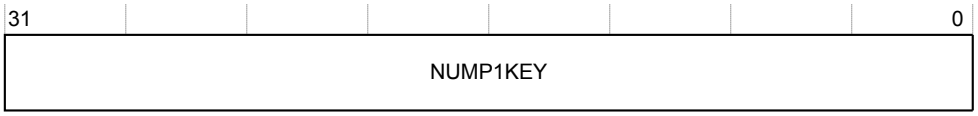


Figure D8-35 TRCIDR10 bit assignments

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`.

D8.38 TRCIDR11, ID Register 11

The TRCIDR11 returns the number of special P1 right-hand keys that the trace unit can use.

Bit field descriptions

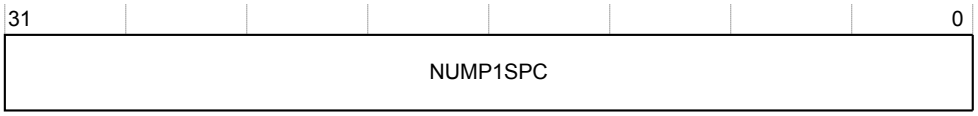


Figure D8-36 TRCIDR11 bit assignments

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`.

D8.39 TRCIDR12, ID Register 12

The TRCIDR12 returns the number of conditional instruction right-hand keys that the trace unit can use.

Bit field descriptions

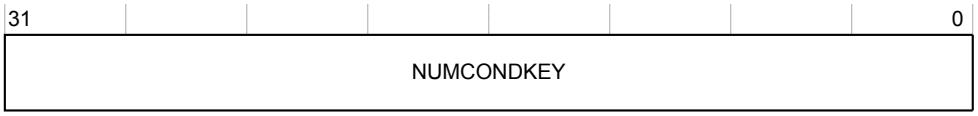


Figure D8-37 TRCIDR12 bit assignments

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`.

D8.40 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

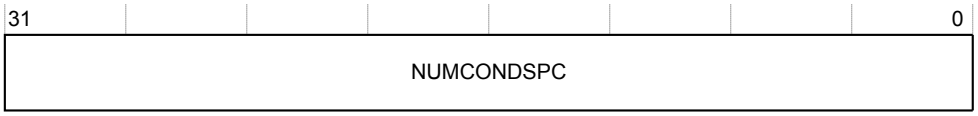


Figure D8-38 TRCIDR13 bit assignments

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`.

D8.41 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

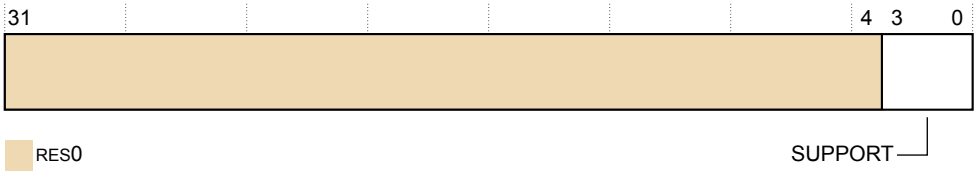


Figure D8-39 TRCIMSPEC0 bit assignments

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`.

D8.42 TRCITATBIDR, Integration ATB Identification Register

The TRCITATBIDR sets the state of output pins mentioned in the bit descriptions in this section.

Bit field descriptions

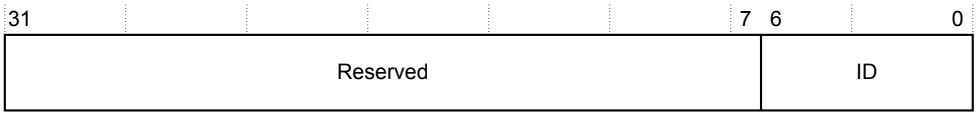


Figure D8-40 TRCITATBIDR bit assignments

[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`.

D8.43 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

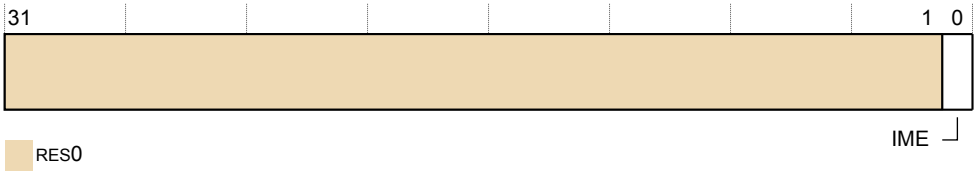


Figure D8-41 TRCITCTRL bit assignments

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`.

D8.44 TRCITIATBINR, Integration Instruction ATB In Register

The TRCITIATBINR reads the state of the input pins described in this section.

Bit field descriptions

Figure D8-42 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`.

D8.45 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 D8-43 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`.

D8.46 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

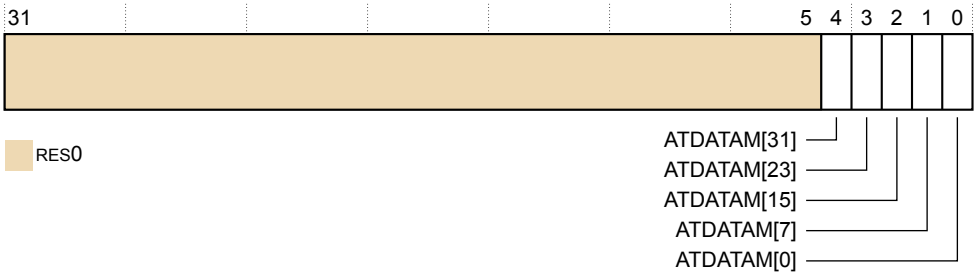


Figure D8-44 TRCITIDATAR bit assignments

RES0, [31:5]

RES0 Reserved.

ATDATAM[31], [4]
Drives the ATDATAM[31] output.[1]

ATDATAM[23], [3]
Drives the ATDATAM[23] output.[1]

ATDATAM[15], [2]
Drives the ATDATAM[15] output.[1]

ATDATAM[7], [1]
Drives the ATDATAM[7] output.[1]

ATDATAM[0], [0]
Drives the ATDATAM[0] output.[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 TRCITIDATAR can be accessed through the external debug interface, offset `0xEEC`.

[1] 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.

D8.47 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

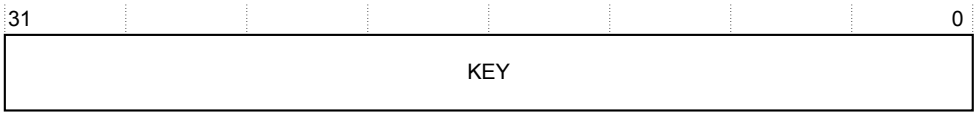


Figure D8-45 TRCLAR bit assignments

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`.

D8.48 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

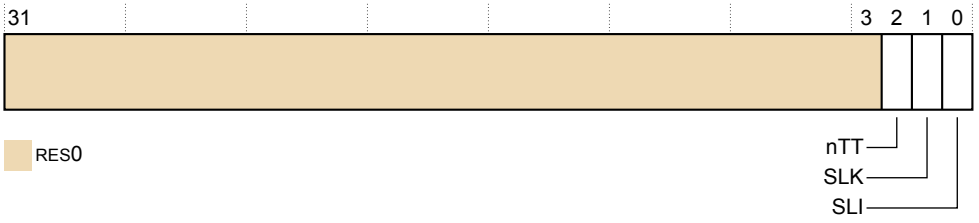


Figure D8-46 TRCLSR bit assignments

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`.

D8.49 TRCCNTVRn, Counter Value Registers 0-1

The TRCCNTVRn contains the current counter value.

Bit field descriptions

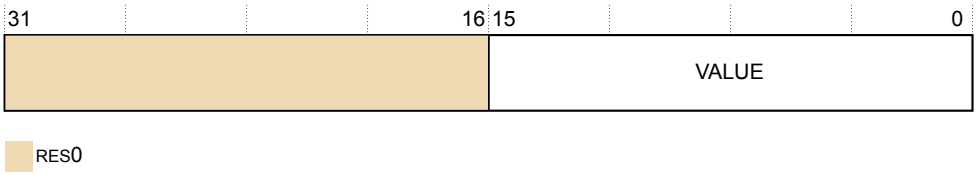


Figure D8-47 TRCCNTVRn bit assignments

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`.

D8.50 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

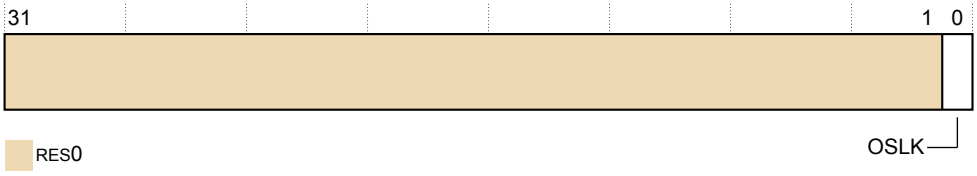


Figure D8-48 TRCOSLAR bit assignments

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`.

D8.51 TRCOSLSR, OS Lock Status Register

The TRCOSLSR returns the status of the OS Lock.

Bit field descriptions

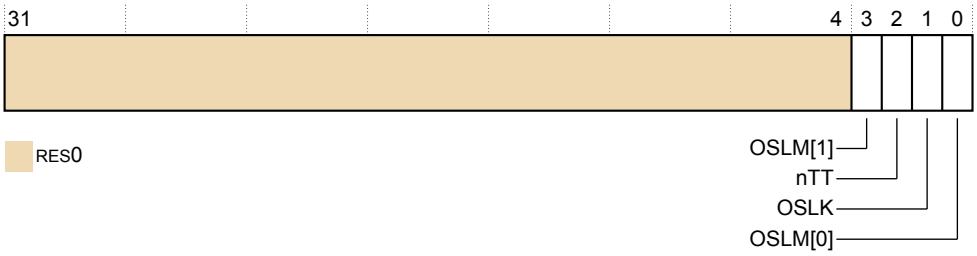


Figure D8-49 TRCOSLSR bit assignments

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`.

D8.52 TRCPDCR, Power Down Control Register

The TRCPDCR request to the system power controller to keep the ETM trace unit powered up.

Bit field descriptions

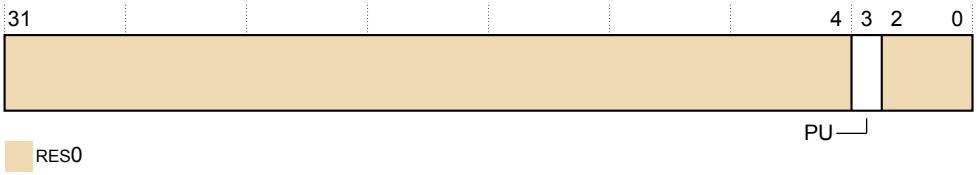


Figure D8-50 TRCPDCR bit assignments

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`.

D8.53 TRCPDSR, Power Down Status Register

The TRCPDSR indicates the power down status of the ETM trace unit.

Bit field descriptions

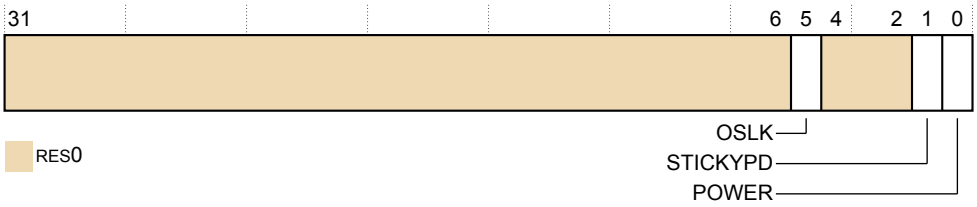


Figure D8-51 TRCPDSR bit assignments

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`.

D8.54 TRCPIDR0, ETM Peripheral Identification Register 0

The TRCPIDR0 provides information to identify a trace component.

Bit field descriptions

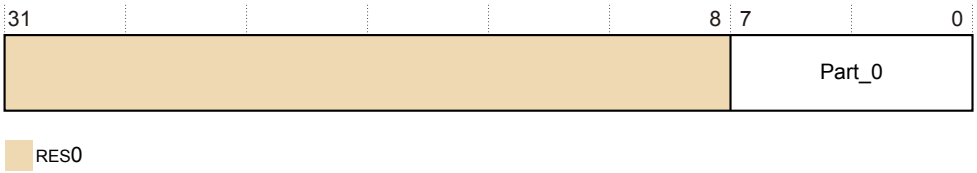


Figure D8-52 TRCPIDR0 bit assignments

RES0, [31:8]

RES0 Reserved.

Part_0, [7:0]

`0x05` 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`.

D8.55 TRCPIDR1, ETM Peripheral Identification Register 1

The TRCPIDR1 provides information to identify a trace component.

Bit field descriptions

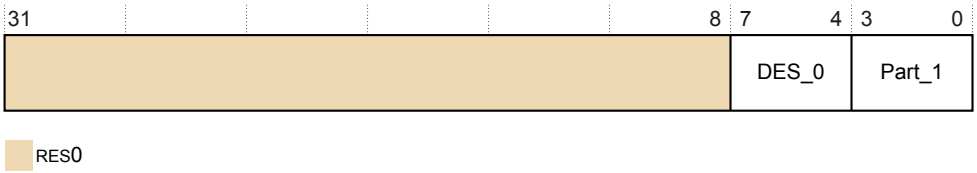


Figure D8-53 TRCPIDR1 bit assignments

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`.

D8.56 TRCPIDR2, ETM Peripheral Identification Register 2

The TRCPIDR2 provides information to identify a trace component.

Bit field descriptions

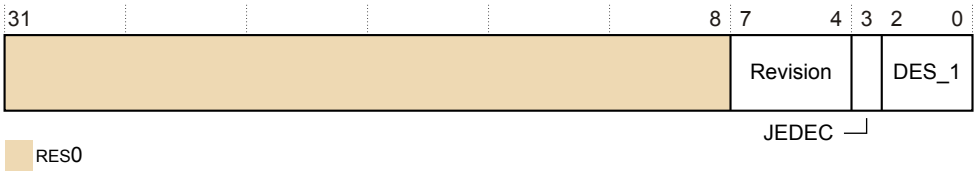


Figure D8-54 TRCPIDR2 bit assignments

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`.

D8.57 TRCPIDR3, ETM Peripheral Identification Register 3

The TRCPIDR3 provides information to identify a trace component.

Bit field descriptions

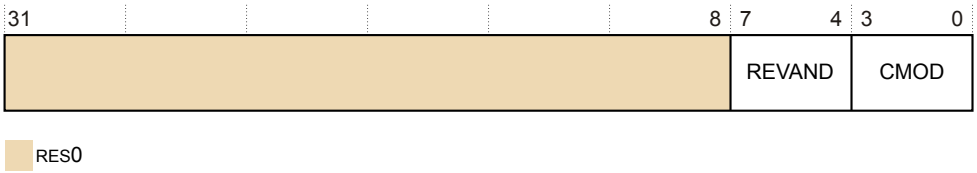


Figure D8-55 TRCPIDR3 bit assignments

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`.

D8.58 TRCPIDR4, ETM Peripheral Identification Register 4

The TRCPIDR4 provides information to identify a trace component.

Bit field descriptions

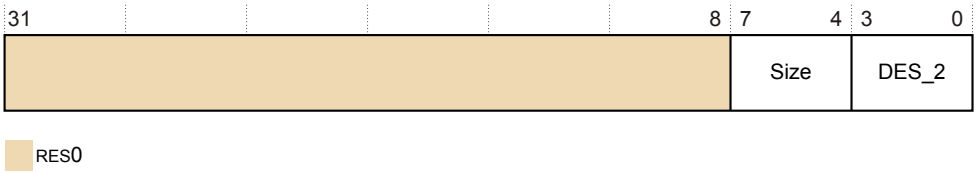


Figure D8-56 TRCPIDR4 bit assignments

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`.

D8.59 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.

D8.60 TRCPRGCTLR, Programming Control Register

The TRCPRGCTLR enables the ETM trace unit.

Bit field descriptions

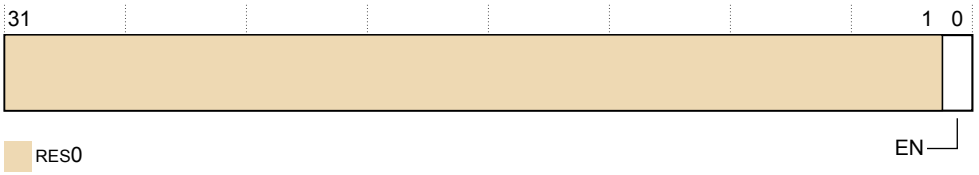


Figure D8-57 TRCPRGCTLR bit assignments

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`.

D8.61 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

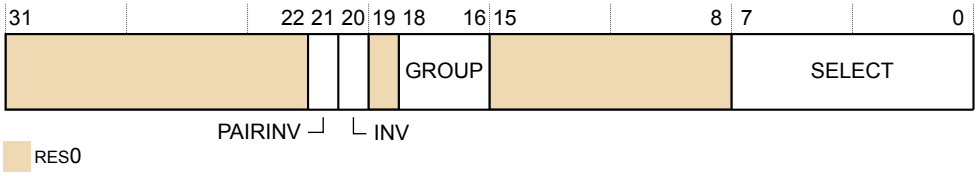


Figure D8-58 TRCRSCTLRn bit assignments

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`.

D8.62 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

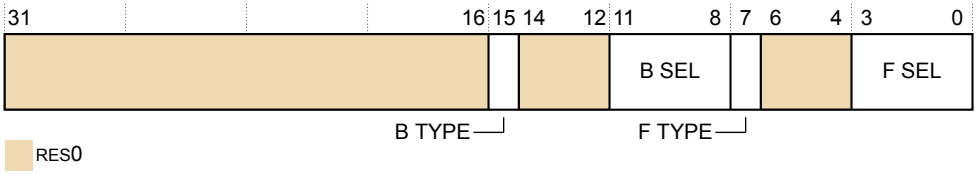


Figure D8-59 TRCSEQEVRn bit assignments

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`.

D8.63 TRCSEQRSTEVR, Sequencer Reset Control Register

The TRCSEQRSTEVR resets the sequencer to state 0.

Bit field descriptions

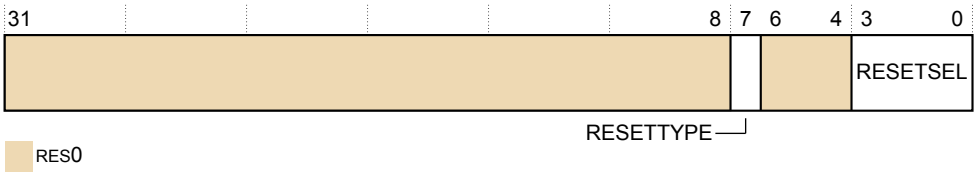


Figure D8-60 TRCSEQRSTEVR bit assignments

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`.

D8.64 TRCSEQSTR, Sequencer State Register

The TRCSEQSTR holds the value of the current state of the sequencer.

Bit field descriptions

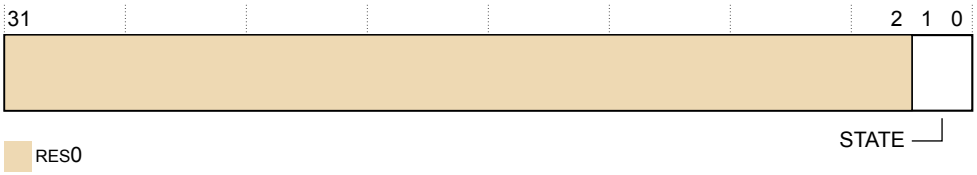


Figure D8-61 TRCSEQSTR bit assignments

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`.

D8.65 TRCSSCCR0, Single-Shot Comparator Control Register 0

The TRCSSCCR0 controls the single-shot comparator.

Bit field descriptions

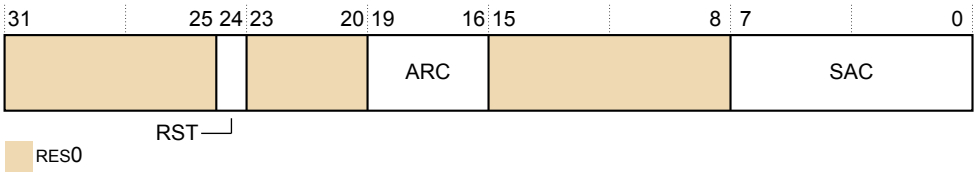


Figure D8-62 TRCSSCCR0 bit assignments

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`.

D8.66 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

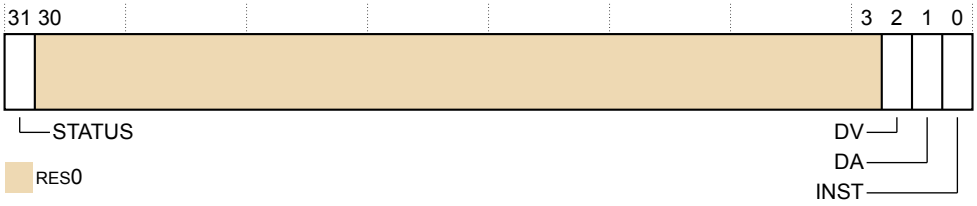


Figure D8-63 TRCSSCSR0 bit assignments

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`.

D8.67 TRCSTALLCTLR, Stall Control Register

The TRCSTALLCTLR enables the ETM trace unit to stall the Cortex-A55 core if the ETM trace unit FIFO overflows.

Bit field descriptions

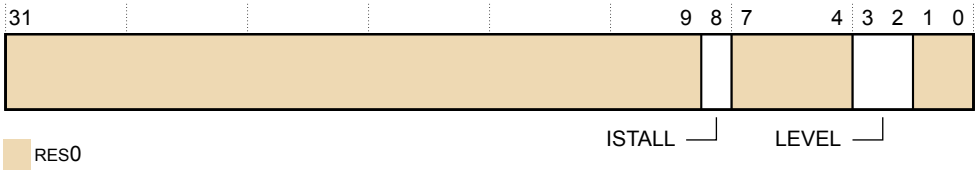


Figure D8-64 TRCSTALLCTLR bit assignments

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`.

D8.68 TRCSTATR, Status Register

The TRCSTATR indicates the ETM trace unit status.

Bit field descriptions

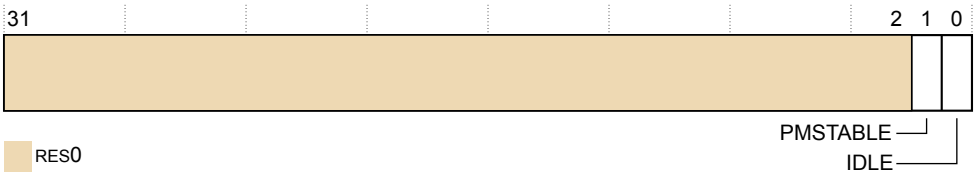


Figure D8-65 TRCSTATR bit assignments

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`.

D8.69 TRCSYNCPR, Synchronization Period Register

The TRCSYNCPR controls how often periodic trace synchronization requests occur.

Bit field descriptions

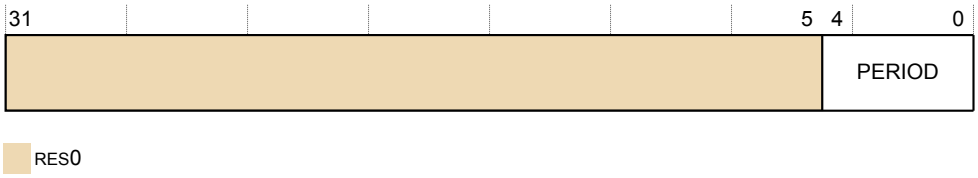


Figure D8-66 TRCSYNCPR bit assignments

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`.

D8.70 TRCTRACEIDR, Trace ID Register

The TRCTRACEIDR sets the trace ID for instruction trace.

Bit field descriptions

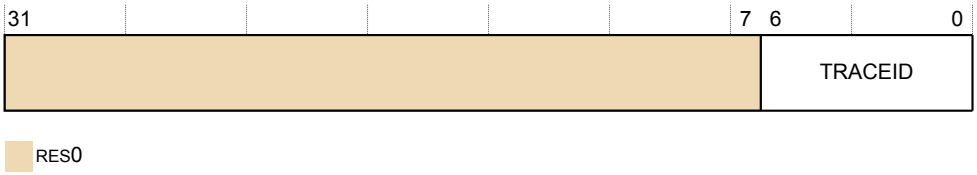


Figure D8-67 TRCTRACEIDR bit Assignments

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`.

D8.71 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

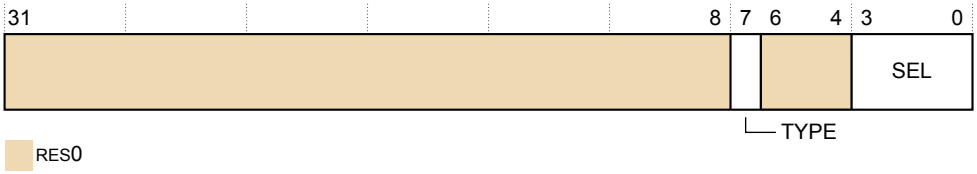


Figure D8-68 TRCTSCTLR bit assignments

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`.

D8.72 TRCVICTLR, ViewInst Main Control Register

The TRCVICTLR controls instruction trace filtering.

Bit field descriptions

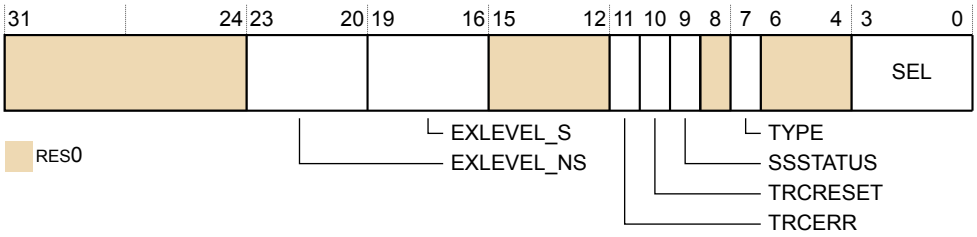


Figure D8-69 TRCVICTLR bit assignments

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`.

D8.73 TRCVIIECTLR, ViewInst Include-Exclude Control Register

The TRCVIIECTLR defines the address range comparators that control the ViewInst Include/Exclude control.

Bit field descriptions

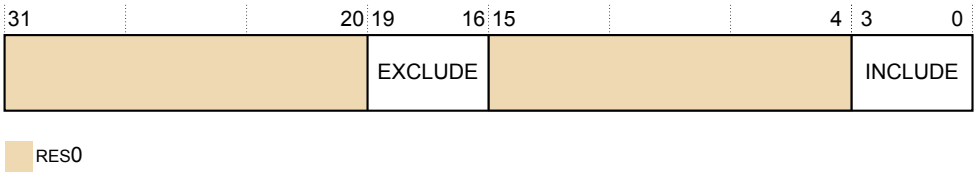


Figure D8-70 TRCVIIECTLR bit assignments

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`.

D8.74 TRCVISSCTLR, ViewInst Start-Stop Control Register

The TRCVISSCTLR defines the single address comparators that control the ViewInst Start/Stop logic.

Bit field descriptions

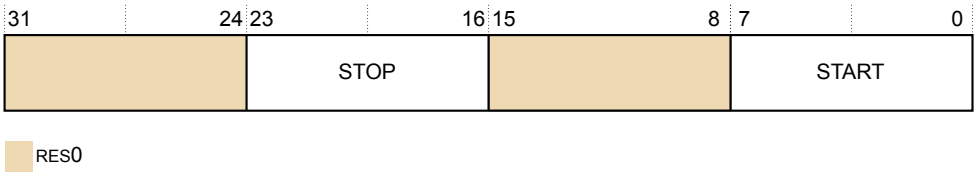


Figure D8-71 TRCVISSCTLR bit assignments

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`.

D8.75 TRCVMIDCVR0, VMID Comparator Value Register 0

The TRCVMIDCVR0 contains a VMID value.

Bit field descriptions

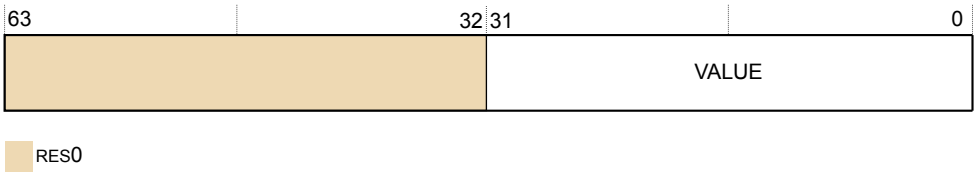


Figure D8-72 TRCVMIDCVR0 bit assignments

RES0, [63:32]

RES0 Reserved.

VALUE, [31:0]

The VMID 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 TRCVMIDCVR0 can be accessed through the external debug interface, offset `0x640`.

Part E
Appendices

Appendix A
AArch32 UNPREDICTABLE Behaviors

This appendix describes the cases in which the Cortex-A55 core implementation diverges from the preferred behavior described in ARMv8 AArch32 UNPREDICTABLE behaviors.

It contains the following sections:

A.1 Use of R15 by Instruction

All uses of R15 as a named register specifier for a source register that is described as UNPREDICTABLE take an UNDEFINED exception trap.

For information on UNPREDICTABLE registers, see the *ARM® Architecture Reference Manual ARMv8, for ARMv8-A architecture profile*.

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-A55 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-A55 processor 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-A55 processor 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

There are unpredictable behaviors associated with Debug.

The information in this section describes which option the Cortex-A55 core implements based on the 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: • Option 3: 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 32-bit instruction. In this case, the breakpoint is not taken.
Address matching breakpoint on A32 instruction with DBGBCRn.BAS=1100	The core implements the following option: • Option 1: Does match.
Address match breakpoint match on T32 instruction at DBGBCRn+2 with DBGBCRn.BAS=1111	The core implements the following option: • Option 1: Does match.
Address mismatch breakpoint match on T32 instruction at DBGBCRn +2 with DBGBCRn.BAS=1111	The core implements the following option: • Option 1: Does match.
Other mismatch breakpoint matches any address in current mode and state	The core implements the following option: • Option 2: Immediate breakpoint debug event.
Mismatch breakpoint on branch to self	The core implements the following option: • Option 2: 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: • Option 1: 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: • Option 1: Does match.
Address-matching Vector Catch on 32-bit T32 instruction at (vector+2)	The core implements the following option: • Option 1: Does match.
Address-matching Vector Catch and Breakpoint on same instruction	The core implements the following option: • Option 2: Report Breakpoint.
Address match breakpoint with DBGBCRn_EL1.BAS=0000	The core implements the following option: • Option 1: As if disabled.
DBGWCRn_EL1.BAS specifies a non-contiguous set of bytes within a doubleword	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: • Option 3: Executed unconditionally.

Table A-1 ARMv8 Debug UNPREDICTABLE behaviors (continued)

Scenario	Behavior
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: • Option 2: 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: • Option 2: 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.
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: • Option 1: The instruction completes in Debug state before executing the restart.

Table A-1 ARMv8 Debug UNPREDICTABLE behaviors (continued)

Scenario	Behavior
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.
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: **Actual behavior:** 1. For reserved debug and Performance Monitors registers the response is CONSTRAINED UNPREDICTABLE Error or RES0, when any of the following error instead of preferred RES0 for reserved debug registers `0x000-0xCFC` and reserved PMU registers `0x000-0xF00`: **Off** 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, OS double-lock is locked, that is, EDPRSR.DLK is 1. **OSLK** OSLSR_EL1.OSLK is1, OS lock is locked. 2. In addition, for reserved debug registers in the address ranges `0x400` to `0x4FC` and `0x800` to `0x8FC`, the response is CONSTRAINED UNPREDICTABLE Error or RES0 when the conditions in *1* do not apply and: **EDAD** `AllowExternalDebugAccess()` is FALSE, external debug access is disabled. 3. For reserved Performance Monitor registers in the address ranges `0x000` to `0x0FC` and `0x400` to `0x47C`, the response is CONSTRAINED UNPREDICTABLE Error, or RES0 when the conditions in *1* and *2* do not apply, and the following errors instead of preferred res0 for the these registers: **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-766.

B.1 Revisions

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

Table B-1 Issue 0000-00

Change	Location	Affects
First release	-	-

Table B-2 Differences between issue 0000-00 and issue 0001-00

Change	Location	Affects
Editorial changes	-	r0p1
Updated the product revision to r0p1	-	r0p1
Minor updates in the components section	*A2.1 Components* on page A2-34	r0p1
Added a set of timer registers	*A2.4 About the Generic Timer* on page A2-39	r0p1
Updated the core dynamic retention mode	*A4.6.5 Core dynamic retention* on page A4-55	r0p1
Updated the section regarding configuring MMU accesses	*A5.5.1 Configuring MMU accesses* on page A5-67	r0p1
Updated the section regarding external aborts	*A5.6.3 External aborts* on page A5-69	r0p1
Updated the section regarding mis-programming contiguous hints	*A5.6.4 Mis-programming contiguous hints* on page A5-69	r0p1
Updated the section regarding conflict aborts	*A5.6.5 Conflict aborts* on page A5-69	r0p1
Updated the direct access to internal memory	*A6.6 Direct access to internal memory* on page A6-82	r0p1
Added information on outstanding simultaneous transactions supported	*A7.1 About the L2 memory system* on page A7-90	r0p1
Updated the support for memory types section	*A7.3 Support for memory types* on page A7-92	r0p1
Updated the cluster registers tables	*B1.3 AArch32 implementation defined register summary* on page B1-121, *B1.4 AArch32 registers by functional group* on page B1-123, *B2.3 AArch64 implementation defined register summary* on page B2-280, *B2.4 AArch64 registers by functional group* on page B2-282	r0p1
Updated the ACTLR_EL2 register	*B2.6 ACTLR_EL2, Auxiliary Control Register, EL2* on page B2-290	r0p1
Updated the ACTLR_EL3 register	*B2.7 ACTLR_EL3, Auxiliary Control Register, EL3* on page B2-292	r0p1
Updated the IFSR32_EL2 register	*B2.75 IFSR32_EL2, Instruction Fault Status Register, EL2* on page B2-394	r0p1
Updated the VDISR_EL2 register at EL1 using AArch64	*B2.98.3 VDISR_EL2 at EL1 using AArch64* on page B2-426	r0p1
Updated the ERR0PFGCDNR register	*B3.7 ERR0PFGCDNR, Error Pseudo Fault Generation Count Down Register* on page B3-444	r0p1
Updated the ERR0PFGCTLR, register	*B3.8 ERR0PFGCTLR, Error Pseudo Fault Generation Control Register* on page B3-445	r0p1
Updated the PMU events	*C2.4 PMU events* on page C2-531	r0p1

Table B-3 Differences between issue 0001-00 and issue 0100-00

Change	Location	Affects
Editorial changes.	-	r1p0
Updated the product revision to r1p0.	-	r1p0
Updated product name.	-	r1p0
Global terminology change from 'processor' to 'core' for the product.	-	r1p0
Updated FCM to DSU.	-	r1p0
Added ELA address size option.	*A1.3 Implementation options* on page A1-28.	r1p0
Updated the encoding for the L2 TLB.	*A6.6.3 Encoding for the L2 TLB* on page A6-84.	r1p0
Added CPU private registers.	*B1.3 AArch32 implementation defined register summary* on page B1-121. *B1.4 AArch32 registers by functional group* on page B1-123. *B1.18 CPUPCR, CPU Private Control Register* on page B1-149. *B1.19 CPUPMR, CPU Private Mask Register* on page B1-151. *B1.20 CPUPOR, CPU Private Operation Register* on page B1-153. *B1.21 CPUPSELR, CPU Private Selection Register* on page B1-155. *B2.3 AArch64 implementation defined register summary* on page B2-280. *B2.4 AArch64 registers by functional group* on page B2-282. *B2.26 CPUPCR_EL3, CPU Private Control Register, EL3* on page B2-318. *B2.27 CPUPMR_EL3, CPU Private Mask Register, EL3* on page B2-320. *B2.28 CPUPOR_EL3, CPU Private Operation Register, EL3* on page B2-322. *B2.29 CPUPSELR_EL3, CPU Private Selection Register, EL3* on page B2-324.	r1p0
Added Dot Product instructions introduced in v8.4.	*A1.3 Implementation options* on page A1-28. *B1.2 AArch32 architectural system register summary* on page B1-115. *B1.4 AArch32 registers by functional group* on page B1-123. *B1.64 ID_ISAR6, Instruction Set Attribute Register 6* on page B1-217. *B2.2 AArch64 architectural system register summary* on page B2-273. *B2.4 AArch64 registers by functional group* on page B2-282. *B2.67 ID_ISAR6_EL1, AArch32 Instruction Set Attribute Register 6, EL1* on page B2-380. *B2.53 ID_AA64ISAR0_EL1, AArch64 Instruction Set Attribute Register 0, EL1* on page B2-356.	r1p0
Updated the CPUECTLR and CPUECTLR_EL1 registers.	*B1.17 CPUECTLR, CPU Extended Control Register* on page B1-146. *B2.25 CPUECTLR_EL1, CPU Extended Control Register, EL1* on page B2-315.	r1p0
Updated the Use of R15 by Instruction.	*A.1 Use of R15 by Instruction* on page Appx-A-758.	r1p0