Arm® DynamIQ™ Shared Unit

Revision: r4p1

Technical Reference Manual

Non-Confidential

Issue 04

100453_0401_04_en

Arm® DynamIQ™ Shared Unit
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
0002-00	20 June 2017	Non-Confidential	First release for r0p2
0100-00	31 May 2017	Confidential	First release for r1p0
0200-00	16 October 2017	Confidential	First release for r2p0
0300-00	26 January 2018	Non-Confidential	First release for r3p0
0300-01	27 April 2018	Non-Confidential	Second release for r3p0
0400-00	27 November 2018	Non-Confidential	First release for r4p0
0400-01	7 December 2018	Non-Confidential	Second release for r4p0
0400-02	14 May 2019	Non-Confidential	Third release for r4p0
0401-03	31 October 2019	Non-Confidential	First release for r4p1
0401-04	30 June 2023	Non-Confidential	Second release for r4p1

Proprietary Notice

Confidentiality Status

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

Unrestricted Access is an Arm internal classification.

Product Status

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

Feedback

Arm welcomes feedback on this product and its documentation. To provide feedback on the product, create a ticket on https://support.developer.arm.com.

To provide feedback on the document, fill the following survey: https://developer.arm.com/documentation-feedback-survey.

Inclusive language commitment

Arm values inclusive communities. Arm recognizes that we and our industry have used language that can be offensive. Arm strives to lead the industry and create change.

This document includes language that can be offensive. We will replace this language in a future issue of this document.

To report offensive language in this document, email terms@arm.com.

Contents

1. Introduction

1.1 Product revision status

The `r`*`x`*`p`*`y`* identifier indicates the revision status of the product described in this manual, for example, `r`*`1`*`p`*`2`*, where:

r*x*	Identifies the major revision of the product, for example, r1.
p*y*	Identifies the minor revision or modification status of the product, for example, p2.

1.2 Intended audience

This manual is written for system designers, system integrators, and programmers who are designing or programming a *System-on-Chip* (SoC) that uses the DSU.

1.3 Conventions

The following subsections describe conventions used in Arm documents.

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: developer.arm.com/glossary.

Convention	Use
italic	Citations.
bold	Terms in descriptive lists, where appropriate.
`monospace`	Text that you can enter at the keyboard, such as commands, file and program names, and source code.
`monospace underline`	A permitted abbreviation for a command or option. You can enter the underlined text instead of the full command or option name.
`<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>`

Convention	Use
SMALL CAPITALS	Terms that have specific technical meanings as defined in the *Arm® Glossary*. For example, IMPLEMENTATION DEFINED, IMPLEMENTATION SPECIFIC, UNKNOWN, and UNPREDICTABLE.

Caution

Recommendations. Not following these recommendations might lead to system failure or damage.

Warning

Requirements for the system. Not following these requirements might result in system failure or damage.

Danger

Requirements for the system. Not following these requirements will result in system failure or damage.

Note

An important piece of information that needs your attention.

Tip

A useful tip that might make it easier, better or faster to perform a task.

Remember

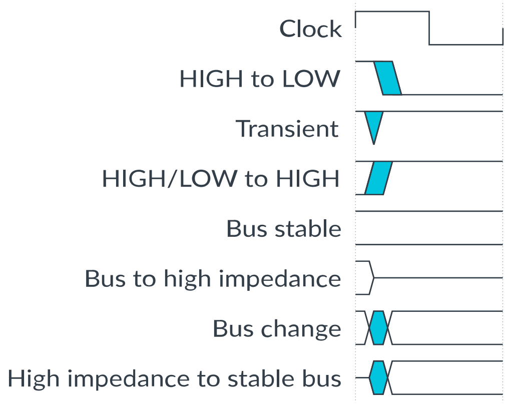
A reminder of something important that relates to the information you are reading.

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 1-1: 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, n denotes an active-LOW signal.

1.4 Useful resources

This document contains information that is specific to this product. See the following resources for other useful information.

Access to Arm documents depends on their confidentiality:

- Non-Confidential documents are available at developer.arm.com/documentation. Each document link in the following tables goes to the online version of the document.
- Confidential documents are available to licensees only through the product package.

Arm product resources	Document ID	Confidentiality
Arm® DynamIQ™ Shared Unit Configuration and Sign-off Guide	*100454*	Confidential
Arm® DynamIQ™ Shared Unit Integration Manual	*100455*	Confidential
Arm® DynamIQ™ Shared Unit Release Note	*ARM-EPM-xxxxxx*	Confidential

Arm architecture and specifications	Document ID	Confidentiality
AMBA® 5 CHI Architecture Specification	*IHI 0050*	Non-Confidential

Arm architecture and specifications	Document ID	Confidentiality
AMBA® APB Protocol Specification	*IHI 0024*	Non-Confidential
AMBA® ATB Protocol Specification	*IHI 0032*	Non-Confidential
AMBA® AXI and ACE Protocol Specification	*IHI 0022*	Non-Confidential
AMBA® Low Power Interface Specification	*IHI 0068*	Non-Confidential
Arm® Architecture Reference Manual for A-profile architecture	*DDI 0487*	Non-Confidential
Arm® CoreSight™ Architecture Specification v3.0	*IHI 0029*	Non-Confidential
Arm® CoreSight™ DAP-Lite2 Technical Reference Manual	*100572*	Non-Confidential
Arm® CoreSight™ ELA-500 Embedded Logic Analyzer Technical Reference Manual	*100127*	Non-Confidential
Arm® CoreSight™ SoC-400 Technical Reference Manual	*DDI 0480*	Non-Confidential
Arm® Embedded Trace Macrocell Architecture Specification ETMv4	*IHI 0064*	Non-Confidential
Arm® Generic Interrupt Controller Architecture Specification, GIC architecture version 3 and version 4	*IHI 0069*	Non-Confidential

Note

Arm tests its PDFs only in Adobe Acrobat and Acrobat Reader. Arm cannot guarantee the quality of its documents when used with any other PDF reader.

Adobe PDF reader products can be downloaded at http://www.adobe.com.

2. The DynamIQ Shared Unit

This chapter introduces the *DynamIQ™ Shared Unit* (DSU) and its features.

2.1 About the DSU

The *DynamIQ™ Shared Unit* (DSU) comprises the L3 memory system, control logic, and external interfaces to support a DynamIQ™ cluster.

The DynamIQ™ cluster microarchitecture integrates one or more cores with the DSU to form a cluster that is implemented to a specified configuration.

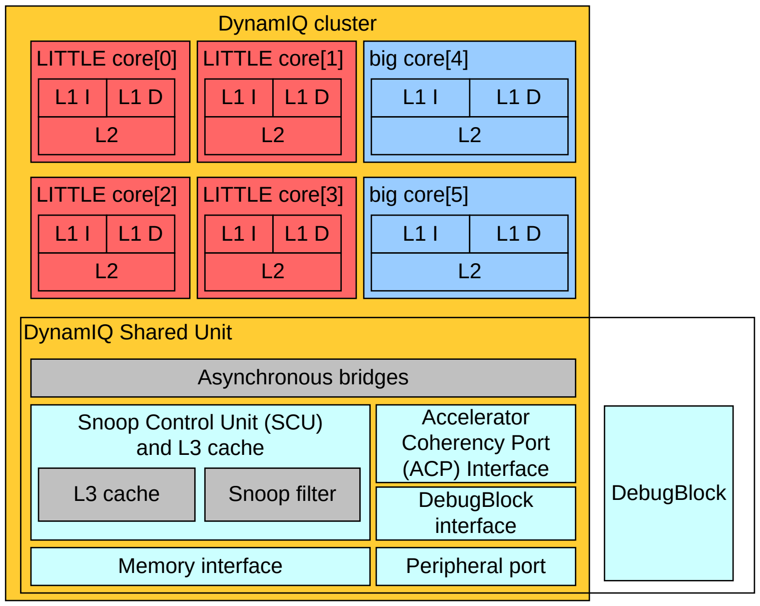
The cores are selected and configured during macrocell implementation.

A cluster can be implemented in one of three configurations:

- A set of cores having the same microarchitecture.
- Two sets of cores, where each set has a different microarchitecture. This configuration might be used in a DynamIQ™ big.LITTLE™ cluster.
- Three sets of cores, where each set has a different microarchitecture.

The following diagram shows a cluster that is composed of two sets of cores in a DynamIQ™ big.LITTLE™ configuration.

Figure 2-1: DynamIQ™ cluster



Within the DSU there is the L3 cache, the *Snoop Control Unit* (SCU), internal interfaces to the cores, and external interfaces to the SoC:

- The shared L3 cache simplifies process migration between the cores.

  Note

  Some cores can be configured without L2 caches. To these cores, the shared L3 cache appears as an L2 cache. The term 'L3 cache' is used throughout this document to describe the shared cache.

- The *Snoop Control Unit* (SCU) maintains coherency between caches in the cores and L3. The SCU includes a snoop filter to optimize coherency maintenance operations.
- Internal interfaces to the cores are configured during macrocell implementation and are not directly visible.
- External interfaces are connected to the SoC.

Each core can be configured either to be run synchronously with the DSU, sharing the clock, or asynchronously, with an independent clock.

Microarchitecture features and system control registers that are specific to the implemented cores are described in separate *Technical Reference Manual* (TRMs) delivered with the cores.

A DebugBlock is provided with the DSU that integrates an Embedded Cross Trigger with debugging registers and supports debug over powerdown. The DebugBlock includes all functionality that is required in the debug power domain.

2.2 Features

The DSU includes the following features:

- AMBA ACE5 or AMBA 5 CHI main bus interface.
- Optional 128-bit wide I/O-coherent *Accelerator Coherency Port* (ACP).
- Optional 64-bit wide device peripheral port.
- Support for cores with 40-bit, 44-bit, or 48-bit physical addresses.
- Arm®v8.2-A debug logic or (for compatible cores) Arm®v8.4-A debug logic.
- *Reliability, Availability, and Serviceability* (RAS) support.
- Optional unified 16-way set-associative L3 cache.
- 64-byte cache lines throughout.
- Cache partitioning support.
- Partial L3 cache powerdown support.
- Optional cache protection in the form of *Error Correcting Code* (ECC) on L3 cache RAM instances.

- *Snoop Control Unit* (SCU).
- L3 memory system can be clocked at a rate synchronous to the external system interconnect or at integer multiples.
- Arm®v8.2-A or Arm®v8.4-A architecture cores supported.
- Support for three types of core.
- Cores can be clocked at different frequencies.

2.3 Implementation options

The DynamIQ™ Shared Unit can be implemented from a range of options. These options are specified during macrocell implementation.

Table 2-1: Configuration parameters

Parameter name	Permitted values	Description
`NUM_BIG_CORES`	0, 1, 2, 3, or 4 cores	The number of big cores that can be implemented depends on the number of LITTLE cores implemented.
`NUM_LITTLE_CORES`	0, 1, 2, 3, 4, 5, 6, 7, or 8 cores	The number of LITTLE cores that can be implemented depends on the number of big cores implemented.
`NUM_OTHER_CORES`	0, 1, 2, or 3 cores	The number of 'other' cores that can be implemented depends on the number of big and LITTLE cores implemented.
`BIG_CORE_TYPE`	-	The type of big core you can implement depends on your license.
`LITTLE_CORE_TYPE`	-	The type of LITTLE core you can implement depends on your license.
`OTHER_CORE_TYPE`	-	The type of 'other' core you can implement depends on your license. **Note:** This `OTHER_CORE_TYPE` is a different big core from the `BIG_CORE_TYPE`.
`MODULE`	-	Name of the cluster top-level Verilog file. If you do not specify any top-level module name, then a name is autogenerated based on the number and types of cores present.
`MODULE_DEBUG_BLOCK`	-	Name of the DebugBlock top-level Verilog file. If you do not specify any DebugBlock module name, then a name is autogenerated based on the top-level module name.
`ACE`	`TRUE`, `FALSE`	Main memory interface. **TRUE** Implement AMBA ACE5. **FALSE** Implement AMBA 5 CHI.
`MASTER_DATA_WIDTH`	128, 256, or 512	Bus width for the main coherent master interface. **128** 1 x 128-bit wide ACE or CHI **256** 2 x 128-bit wide ACE, or 1x 256-bit wide CHI **512** 2 x 256-bit wide CHI
`PORTER_SAM`	`TRUE`, `FALSE`	*System Address Map* (SAM). Only applicable if the CHI interface is implemented. **TRUE** Include support for the CMN-600 interconnect SAM. **FALSE** Do not include.

Parameter name	Permitted values	Description
`ACP`	`TRUE`, `FALSE`	*Accelerator Coherency Port* (ACP) **TRUE** Include ACP. **FALSE** Do not include ACP.
`PERIPH_PORT`	`TRUE`, `FALSE`	Peripheral port **TRUE** Include Peripheral port. **FALSE** Do not include Peripheral port.
`SCU_CACHE_PROTECTION`	`TRUE`, `FALSE`	*Error Correcting Code* (ECC) support **TRUE** Protect the L3 cache RAMs and snoop filter RAMs with ECC. **FALSE** No ECC on L3 cache RAMs and snoop filter RAMs. **Note:** The DSU ECC protection implementation is configured in conjunction with the cache protection parameter of your core (`CORE_CACHE_PROTECTION`). See the documentation for your core for more information. The valid combinations are: • `CORE_CACHE_PROTECTION` = `FALSE`, `SCU_CACHE_PROTECTION` = `FALSE`. • `CORE_CACHE_PROTECTION` = `FALSE`, `SCU_CACHE_PROTECTION` = `TRUE`. • `CORE_CACHE_PROTECTION` = `TRUE`, `SCU_CACHE_PROTECTION` = `TRUE`.
`L3_CACHE`	`TRUE`, `FALSE`	L3 cache present **TRUE** Include L3 cache. **FALSE** Do not include L3 cache.
`L3_CACHE_SIZE`	256KB 512KB 1024KB 1536KB 2048KB 3072KB 4096KB	L3 cache size **Note:** • Only some cores support 256KB. • Software reports a cache size of: 2048KB when the 1536KB size is enabled and 4096KB when the 3072KB size is enabled. Since the non-power-of-two cache size is implemented as a power-of-two size with a quarter of the ways that are permanently powered down.

Parameter name	Permitted values	Description
`NUM_SLICES`	1 or 2	Optional configuration for the number of L3 cache slices. See 6.7 Cache slices and portions on page 62 for more details on cache slices. If this parameter is not provided, then the default number of slices is used, based on the other configuration options. This parameter may be used to override the default number of slices if all of the following are true: • The L3 cache size is configured as 256KB, 512KB or 1024KB, or no L3 cache is configured. • There is only a single master port. If a larger cache size or two master ports are configured, then only 2 slices are supported.
`L3_DATA_WR_LATENCY`	1, 2, or 2p	L3 cache data RAM input latency: • 1 cycle latency. • 2 cycles latency. • 2p, 2 cycle latency with an extra 1 cycle delay, which limits writes to one every 3 cycles. When `L3_DATA_WR_LATENCY` is set to 2p, the `L3_DATA_RD_LATENCY` must be set to 3.
`L3_DATA_RD_LATENCY`	2 or 3	L3 cache data RAM output latency: • 2 cycles latency. • 3 cycles latency. **Note:** `L3_DATA_RD_LATENCY` must be set to 3 when `L3_DATA_WR_LATENCY` is set to 2p.
`L3_DATA_RD_SLICE`	`TRUE`, `FALSE`	L3 cache data RAM output register slice **TRUE** Include L3 data RAMs output register slice. **FALSE** Do not include register slice.
`L3_DATA_STRETCH_CLK`	`TRUE`, `FALSE`	Stretch the clock to the L3 data RAMs. **TRUE** Stretch the clock so that the RAM clock pulse is HIGH for a whole SCLK cycle. **FALSE** Use only gating, so that the RAM clock pulse is HIGH for half an SCLK cycle. **Note:** In either case, the L3 RAM will not be accessed on consecutive SCLK cycles. This option has no performance impact.
`CORE_REG_SLICE`	`TRUE`, `FALSE`	For each core, include a register slice between the core and the *Snoop Control Unit* (SCU). For each core present, starting with core 0, set the value to `TRUE` or `FALSE`. **TRUE** Include a register slice between the core and the SCU. **FALSE** Do not include a register slice.

Parameter name	Permitted values	Description
`ASYNC_BRIDGE`	`TRUE`, `FALSE`	For each core, asynchronous bridge on core to L3 coherent interface. For each core present, starting with core 0, set the value to `TRUE` or `FALSE`. **TRUE** Include an asynchronous bridge for the main bus between core and L3. **FALSE** Do not include an asynchronous bridge. **Note:** The choice affects the internal interface that is used for instruction and data fetch, evictions, and snoops. It does not affect the other interfaces such as debug, trace, and GIC which are always asynchronous.
`INTERLEAVE_ADDR_BIT`	6, 7, 8, 9, 10, 11, or 12	Controls which physical address bit is used to interleave requests between cache slices and dual ACE or dual CHI masters. The default value is bit 6, which interleaves on cache line boundaries. Other values can only be configured when dual ACE or dual CHI interfaces are configured. **Note:** Interleaving on a larger granularity might help improve system performance on some SoC designs. However, it can also reduce performance under some circumstances on accesses that hit in the L3 cache, because the same interleave is used for both cache slices and ACE master. Therefore if changing this parameter, Arm® recommends performing benchmarking in your system to determine if the overall performance is acceptable.
`CORE_SYNC_LEVELS`	2 or 3	Number of synchronizer stages in all asynchronous inputs into the core.
`SYNC_LEVELS`	2 or 3	Number of synchronizer stages in all asynchronous inputs to the SCU and cluster logic.
`LEGACY_V7_DEBUG_MAP`	`TRUE`, `FALSE`	Legacy v7 debug memory map. Configure v7 or v8 Debug memory map. **TRUE** v7 Debug memory map. **FALSE** v8 Debug memory map.
`ELA`	`TRUE`, `FALSE`	Support for integrating the CoreSight ELA-500 Embedded Logic Analyzer. **TRUE** Include one ELA-500 instance within the DSU. **FALSE** Do not include the ELA-500 within the DSU. **Note:** If enabled, to create a unique ELA instance for the DSU, either: • Use the `-ela` option with the generate script. • Run the uniquify script that is supplied with the ELA. For more information, see *Configuring the execution testbench with the ELA-500* in the *Arm® DynamIQ™ Shared Unit Configuration and Sign-off Guide*.
`ELA_RAM_ADDR_SIZE`	2-25	The number of index bits in the ELA-500 RAM. For more information, see *Configuration parameters* section in the *Arm® DynamIQ™ Shared Unit Configuration and Sign-off Guide*.

Parameter name	Permitted values	Description
`L3_PBHA`	`TRUE,FALSE`	Store the *Page-Based Hardware Attributes* (PBHA) in the L3 cache. **TRUE** PBHA bits are stored in the L3 cache so are valid for L3 evictions to the master port. **FALSE** PBHA bits are not stored in the L3 cache so are only valid for read transactions to the master port. **Note:** Not all cores provide PBHA bits. Only enable `L3_PBHA` when all cores support the PBHA bits. For more information about the PBHA bits, see 3.4 Page-based hardware attributes on page 31.
`POP_RAM`	`TRUE,` `FALSE`	Configure the RTL for Arm *Processor Optimization Pack* (POP) RAMs. **TRUE** Configure the RTL for Arm POP RAMs. **FALSE** Do not configure the RTL for Arm POP RAMs. **Note:** The RAM interface is different when POP_RAM is set to `TRUE`. Only set `POP_RAM` to `TRUE` when using Arm POP RAMs.

2.3.1 Cluster configurations

A cluster can be created with one or more cores of the same type.

Alternatively, the cluster can be configured as a big.LITTLE cluster, with two different types of core in the same cluster.

You can also configure a cluster with three types of core in the same cluster, where the third type of core is designated as other. This other core can be any type of big core.

For information on the number and type of permissible cores in the DSU, see the appendix *Compatible core versions* in the *Arm® DynamIQ™ Shared Unit Configuration and Sign-off Guide*.

2.4 Supported standards and specifications

The DynamIQ™ Shared Unit complies with the Arm®v8.2-A or Arm®v8.4-A architecture.

Table 2-2: Compliance with standards and specifications

Architecture specification or standard	Version
Arm architecture	Arm®v8.2-A or Arm®v8.4-A depending on core.
Advanced Microcontroller Bus Architecture (AMBA)	• AMBA ACE5 (Issue F.b). • AMBA 5 CHI (Issue B or Issue C, depending on configured cores).
CoreSight™	v3.

Architecture specification or standard	Version
Debug	Arm®v8.2-A or Arm®v8.4-A depending on core.
Generic Interrupt Controller(GIC) architecture CPU interface and Stream Protocol interface.	v4.0 **Note:** The DSU uses Affinity Level-1 to distinguish between different cores, which some interrupt controllers, such as GIC-500 do not support.
Performance Monitoring Unit (PMU)	PMUv3.

2.5 Test features

The DSU provides interfaces for manufacturing test.

The following manufacturing test interfaces are supported:

DFT	For logic testing.
MBIST	For RAM testing.

2.6 Design tasks

The DynamIQ™ Shared Unit is delivered as a synthesizable *Register Transfer Level* (RTL) description in SystemVerilog HDL. Before you can use it, you must implement, integrate, 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 DSU and its associated cores.

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

The system programmer develops the software to configure and initialize the DSU and its associated cores 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 DSU 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 DSU by programming particular values into registers. The configuration choices affect the behavior of the DSU and its associated cores.

2.7 Product revisions

This section describes the differences in functionality between product revisions.

r0p0	First release.
r0p1	CHI and ACP support added.
r0p2	There are no functional changes in this release.
r1p0	Support for no L3 cache, improved cache stashing support, cache partitioning support, and additional configuration options.
r2p0	Support added for three types of core, CHI.C for Direct connect, and 1.5MB L3 cache size.
r3p0	Support added for dual CHI masters.
r4p0	Support added for CHI.D for Direct connect, 256kB cache size, and transport of *Page-Based Hardware Attributes* (PBHA).
r4p1	Support added for additional core combinations, such as: 5 LITTLE cores; 4 big and 2 LITTLE cores, and transport of CHI REQSRCATTR signals for Direct connect, when required.

3. Technical overview

This chapter describes the structure of the DSU.

3.1 Components

A DynamIQ™ cluster system comprises of two top-level modules:

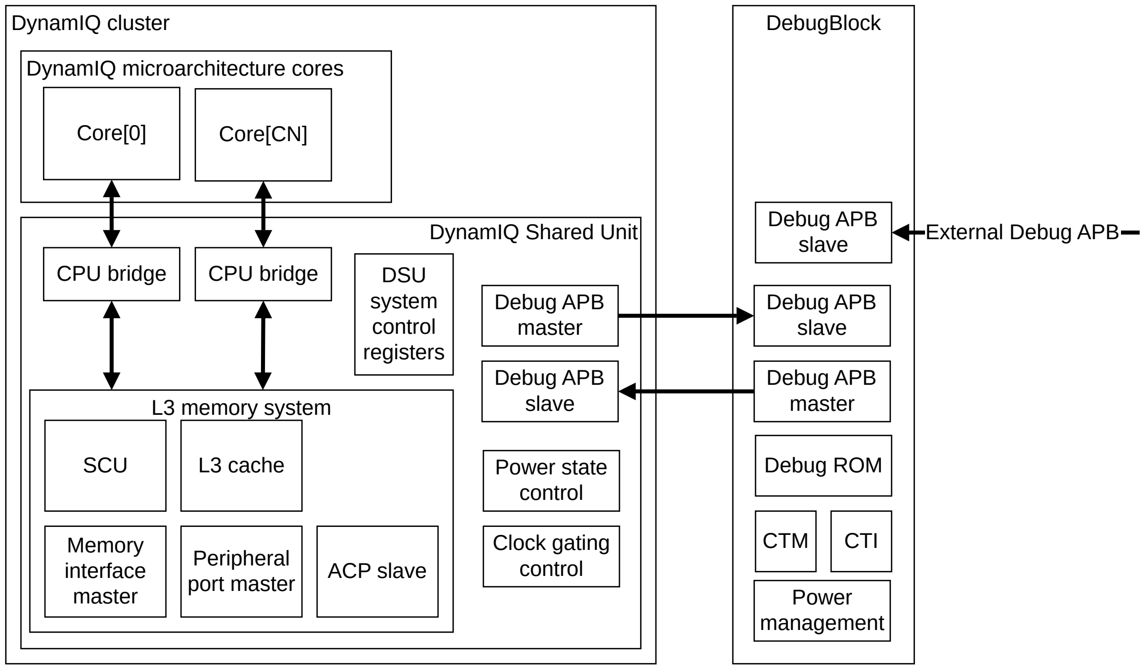
- A module which includes the cores and the *DynamIQ™ Shared Unit* (DSU).
- The DebugBlock.

In this book, the DynamIQ™ cluster is referred to as the cluster.

Separating the debug components from the cluster allows the debug components to be implemented in a separate power domain, allowing debug over power down.

The following figure shows the major components in a DynamIQ™ cluster system.

Figure 3-1: DynamIQ™ cluster components



Throughout this book CN represents a set of cores, where CN has a value of the total number of cores -1. The Arm architecture allows for cores to be single, or multithreaded. A *Processing Element* (PE) performs a thread of execution. A single-threaded core has one PE and a multithreaded core has two or more PEs. Where a reference to a core is made, the core can be a single, or multithreaded core. Signal names that are associated with PEs use the abbreviation PE, where PE has a value of the total number of PEs - 1.

DynamIQ™ cluster compatible cores

The core types, and optional features are selected by the macrocell implementer. The cores are described in their respective TRMs. Because the cores are instantiated within the cluster, all interfacing between the cores and the DSU is implemented automatically. All external signal inputs and outputs pass through the DSU. The DSU buffers and resynchronizes many of these signals to allow the cores to be clocked at different speeds. The memory interface of each core is internally connected to the DSU L3 memory system. Where necessary, the DSU implements additional buffering to compensate for different clock rates of the core and DSU L3. Each core implements clock and power control interfaces. These interfaces are routed through the DSU to the respective core external power controller.

CPU bridges

The CPU bridges control buffering and synchronization between the cores and DSU.

Note

The CPU bridge RTL is included with the DSU deliverables, but is instantiated in the hierarchy of processor cores.

Snoop Control Unit (SCU)

The SCU maintains coherency between all the data caches in the cluster.

The SCU contains buffers that can handle direct cache-to-cache transfers between cores without having to read or write data to the L3 cache. Cache line migration enables dirty cache lines to be moved between cores. Also, there is no requirement to write back transferred cache line data to the L3 cache.

Clock and power management

The cluster supports a set of power-saving modes that are controlled by an external power controller. The modes are selected through power-mode requests on P-Channels, for each of the cores, and a separate P-Channel for the DSU.

Clock gating is supported through Q-Channel requests from an external clock controller to the DSU. The Q-Channels allow individual control of the SCLK, PCLK, ATCLK, and GICCLK clock inputs.

L3 memory interfaces

Main memory master

The main memory interface supports up to two ACE or CHI master interfaces.

Accelerator Coherency Port

The *Accelerator Coherency Port* (ACP) is an optional slave interface. The ACP provides direct memory access to cacheable memory. The SCU maintains cache coherency by checking ACP accesses for allocation in the core and L3 caches. The ACP implements a subset of the ACE-Lite protocol.

Peripheral port

The peripheral port is an optional master interface and provides **DEVICE** accesses to tightly coupled accelerators. The port implements the AXI 4 master interface protocol.

L3 cache

The cache size is implemented as either 256KB, 512KB, 1MB, 1.5MB, 2MB, 3MB, or 4MB. The 1.5MB and 3MB caches are 12-way set associative. The 256KB, 512KB, 1MB, 2MB, and 4MB caches are 16-way set associative. All caches have a 64-byte line length. ECC protection of data and tag RAMs is optionally implemented.

DSU system control registers

The DSU implements a set of system control registers, which are common to all cores in the cluster. You can access these registers from any core in the cluster. These registers provide:

- Control for power management of the cluster.
- L3 cache partitioning control.
- CHI QoS bus control and scheme ID assignment.
- Information about the hardware configuration of the DSU.
- L3 cache hit and miss count information.

Debug and trace components

Each core includes an *Embedded Trace Macrocell* (ETM) to allow program tracing while debugging.

Trigger events from the cores are combined and output to the Debug APB master. Trigger events to the cores, and debug register accesses, are received on the Debug APB slave.

DebugBlock

The debug components are split between two architecturally defined logical power domains, the core power domain, and the debug power domain:

- The core power domain comprises one or more physical power domains for the cores and DSU.
- The debug power domain includes the DebugBlock.

The separate power domains allow the cores and the cluster to be powered down while maintaining essential state that is required to continue debugging. Separating the logical power domains into physical domains is optional and might not be available in individual systems.

Cluster to DebugBlock APB

Trigger events from the cores are transferred to the DebugBlock as APB writes.

DebugBlock to cluster APB

Trigger events to the cores are transferred as APB writes to the DSU. Register accesses from the system debug APB are transferred to the DSU.

System debug APB

The system debug APB slave interface connects to external CoreSight™ components, such as the *Debug Access Port* (DAP).

CTI and CTM

The DebugBlock implements an *Embedded Cross Trigger* (ECT). A *Cross Trigger Interface* (CTI) is allocated to each PE in the cluster, and an additional CTI is allocated to the cluster ELA when present. The CTIs are interconnected through the *Cross Trigger Matrix* (CTM). A single external channel interface is implemented to allow cross-triggering to be extended to the SoC.

Debug ROM

The ROM table contains a list of components in the system. Debuggers can use the ROM table to determine which CoreSight components are implemented.

Power management and clock gating

The DebugBlock implements two Q-Channel interfaces, one for requests to gate the PCLK clock, and a second for requests to control the debug power domain.

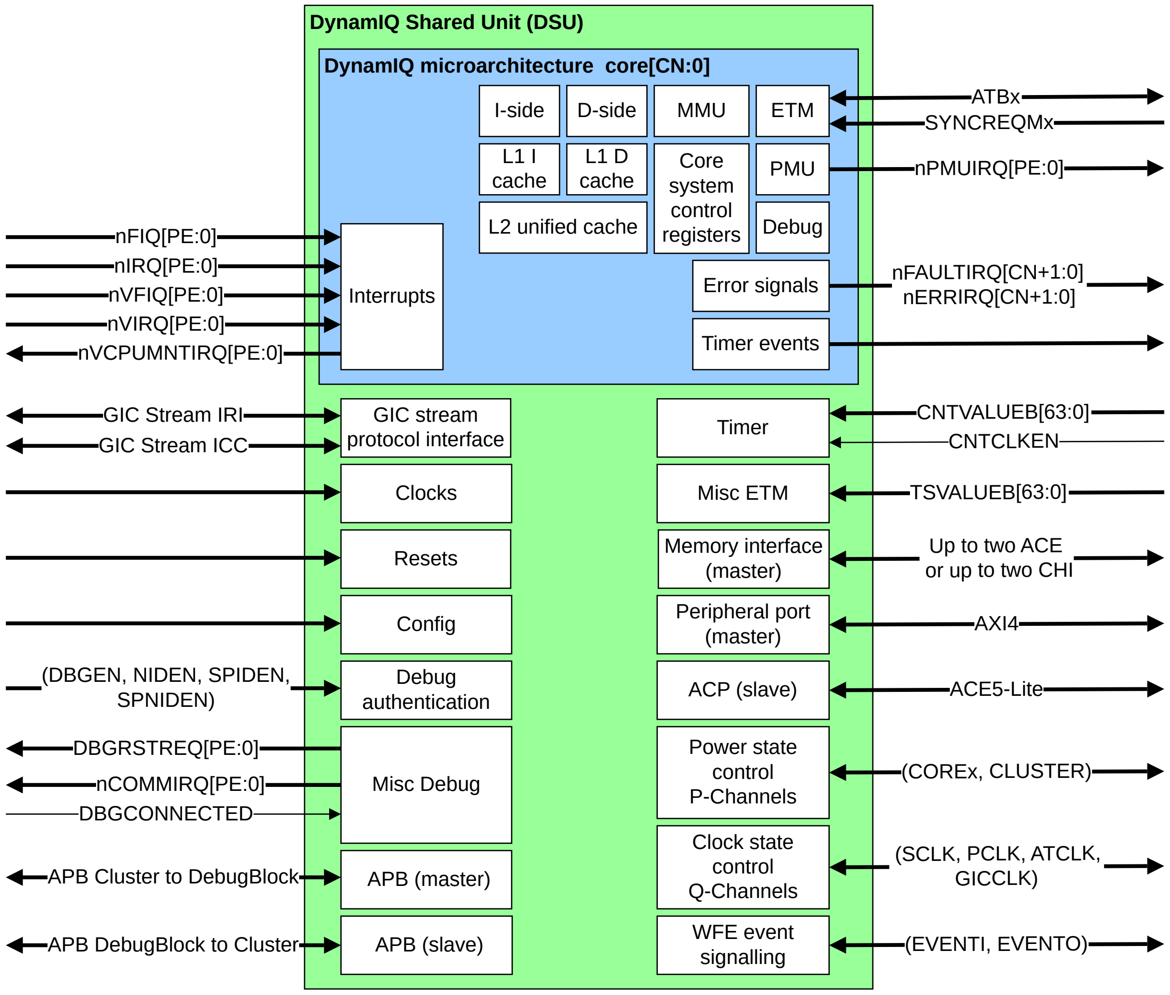
3.2 Interfaces

The external interfaces to connect to the SoC system.

DSU interfaces

The following figure shows the major external interfaces of the *DynamIQ™ Shared Unit* (DSU).

Figure 3-2: DSU interfaces



The following table describes the major external interfaces.

Table 3-1: DSU interfaces

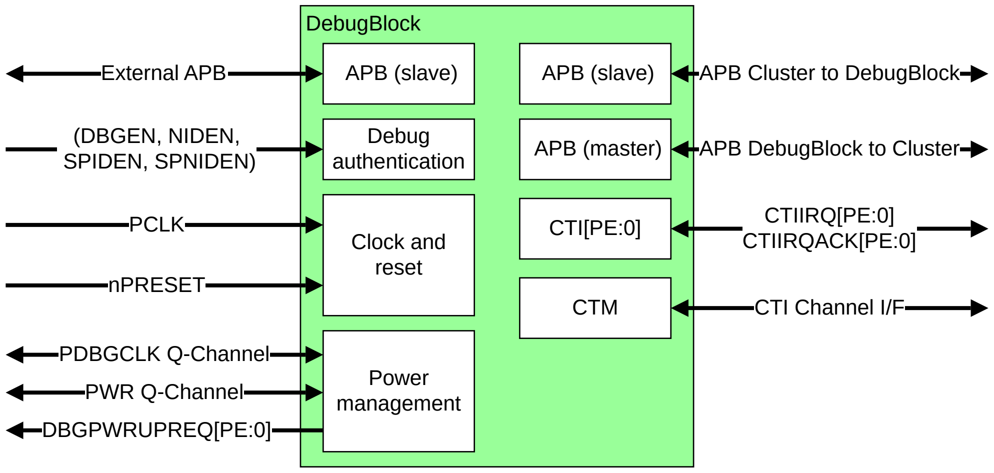
Purpose	Protocol	Notes
Trace	ATB	Master ATB interfaces. Each core has an ATB interface to output ETM trace information, see Appendix B.2.14 ATB Interface Signals.
Memory	ACE or CHI	Master interface to main memory. You can configure the DSU with either 1 or 2 ACE, or 1 or 2 CHI interfaces. See 2.3 Implementation options on page 18.
Accelerator Coherency Port (optional)	ACE5-Lite	Slave interface allowing an external master to make coherent requests to cacheable memory.
Peripheral port (optional)	AXI	Low-latency master interface to external Device memory.
Cluster to DebugBlock	APB	APB interface from the cluster (master) to the DebugBlock (slave).
DebugBlock to cluster	APB	APB interface from the DebugBlock (master) to the cluster (slave).

Purpose	Protocol	Notes
Power state control	P-Channel	P-Channels for DSU and core power management.
Clock state control	Q-Channel	Q-Channels for clock gating control.
WFE event signaling	-	Signals for *Wait For Event* (WFE) wake up events.
Generic timer	-	Input for the generic timer count value. The count value is distributed to all cores. Each core outputs timer events.
GIC interfaces	-	Interrupts to individual cores. A single GIC Stream Protocol interface is shared by all cores.
Design for Test (DFT)	-	Interface to allow access for *Automatic Test Pattern Generation* (ATPG) scan-path testing.
Memory Built-In Self Test (MBIST)	Arm MBIST	Internal interface that supports the manufacturing test of the L3 and SCU memories embedded in the DSU. Each core has its own internal MBIST interface.

DebugBlock interfaces

The following figure shows the major external interfaces.

Figure 3-3: DebugBlock interfaces



The following table describes the major external interfaces.

Table 3-2: DebugBlock interfaces

Purpose	Protocol	Notes
External debug	APB	Slave interface to external debug component, for example a *Debug Access Port* (DAP). Allows access to Debug registers and resources.
Cluster to DebugBlock	APB	APB interface from the cluster (master) to the DebugBlock (slave).
DebugBlock to cluster	APB	APB interface from the DebugBlock (master) to the cluster (slave).
Cross-trigger channel interface	CTI	Allows cross-triggering to be extended to external SoC components.
Power management	Q-Channel	Enables communication to an external power controller. To control clock gating and powerdown.

3.3 RAS support

The DSU supports *Reliability, Availability, and Serviceability* (RAS) features.

- Data poisoning on a 64-bit granule is supported in the DSU.
- Optional ECC protection on RAMs.
- Error recovery and fault handling interrupt outputs.
- Error record registers.

3.4 Page-based hardware attributes

The *Page-Based Hardware Attributes* (PBHA) bits are provided by the processor cores, and passed on or preserved by the DSU.

The PBHA bits are provided externally as part of the SRCATTR bus. PBHA affects the following:

RAM sizes

To generate accurate PBHA bits on L3 cache evictions, store the bits in the L3 cache, by setting the `L3_PBHA` parameter. If this parameter is not set, the PBHA bits are only accurate for read transactions. If this parameter is set, the width of all L3 tag RAM instances is increased by two bits.

ACP

All requests from ACP are given a fixed PBHA value of `0b00`.

Cache stash transactions on CHI

Cache stash transactions might be sent on the CHI interface. For these requests, the PBHA bits being used must be sent along with the stash snoop transaction. The RXSNPFLIT[FwdTxnID] field has two spare bits which are used for this functionality, FwdTxnID[7:6]. If the interconnect does not support PBHA, then these bits are driven to `0b00` by default.

Transaction support

Transactions that do not have a physical address that is associated with them, for example DVM messages, do not provide the PBHA bits. Evict transactions that do not provide any data, for use in de-allocating a snoop filter, do not provide PBHA bits.

Mismatched aliases

If the same physical address is accessed through more than one virtual address mapping, and the PBHA bits are different in the mappings, then the results are **UNPREDICTABLE**. The PBHA value sent on the bus could be for either mapping.

Note

Not all cores provide the PBHA bits. When used with a core that does not provide the PBHA bits, the PBHA output of the DSU is not valid.

3.5 L3 memory system variants

The L3 cache might not be implemented in your system. There are three possible variants.

Possible L3 memory system implementations are:

L3 cache present

This implementation is the default. It provides the most functionality and is suitable for general-purpose workloads.

L3 cache not present

The L3 cache is not present, but snoop filter and SCU logic are present.

This variant allows multiple cores in the cluster, and manages the coherency between them. It supports other implementation options such as ACP, peripheral port, and ACE or CHI master ports. There is an area saving from not including the L3 cache RAMs, however performance of typical workloads is reduced. Therefore Arm recommends that this variant is only used in specialized use cases, or when there is a system cache present that the cores can use.

Direct connect

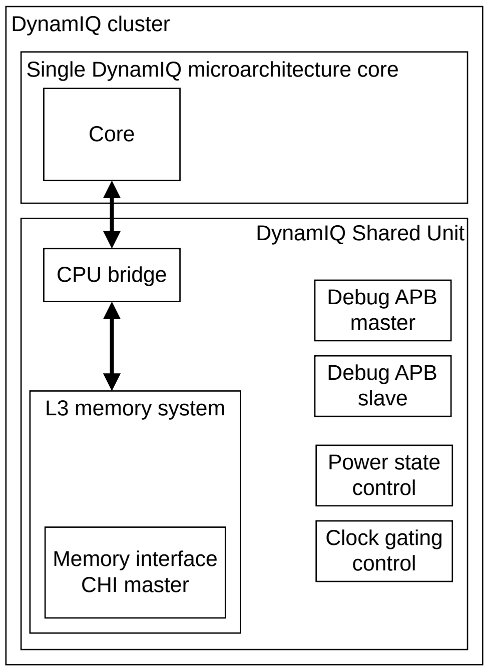
The L3 cache, snoop filter, and SCU logic are not present.

This variant is specifically for use with a CHI interconnect. It offers additional area savings and reduced latency when compared to the previous variants. Because there is no L3 cache in the cluster, this variant relies on the system cache in the interconnect for performance. Only some types of core support this variant. Of the cores that do support this variant, some only support this variant when used with the CMN-600 interconnect as they integrate its system address map component, while others do not have this restriction. See the Technical Reference Manual of your core to identify if your core supports this variant.

Because this variant does not include any coherency logic, it is only supported when there is a single core in the cluster. Optional interfaces such as ACP or the peripheral port are not supported. See 2.3 Implementation options on page 18 for more information. The master port must be a single 256-bit CHI interface.

The following diagram shows the DSU implemented with L3 Direct connect.

Figure 3-4: L3 Direct connect implementation



4. Clocks and resets

This chapter describes the clocks and resets of the DSU.

4.1 Clocks

The DSU requires clock signals for each of the cores, internal logic, and external interfaces.

The following table describes the clocks.

Table 4-1: DSU clock signals

Signal	Description
CORECLK[CN:0]	The per-core clocks for all core logic including L1 caches and L2 caches.
SCLK	The clock for the SCU and L3 memory system, including the ACE or CHI master interface. SCLK is also used for any cores that are configured to run synchronously to the DSU.
PCLK	The clock for the DebugBlock and DSU debug APB interfaces. **Note:** The DebugBlock and cluster both have PCLK inputs. You might choose to connect these to the same clock. Alternatively, you might choose to place an asynchronous bridge between the two clock inputs, in which case they might be different clocks.
ATCLK	The clock for the ATB trace buses output from the DSU. **Note:** All ATB buses output from the DSU share the same clock.
GICCLK	The clock for the GIC AXI-stream interface between the DSU and an external GIC.
PERIPHCLK	The clock for peripheral logic inside the DSU such as timers, and clock and power management logic.

All clocks can be driven fully asynchronously to each other. The DSU contains all the necessary synchronizing logic for crossing between clock domains. There are no clock dividers and no latches in the design. The entire design is rising edge triggered.

Note

- It is possible to configure some or all the cores to run synchronously with the L3 memory system. If this option is chosen, the corresponding CORECLK signals are not present and the synchronous cores run with SCLK.
- The DebugBlock can be clocked by a different clock from the DSU PCLK. To allow this, the macrocell implementer can choose to add asynchronous bridges between the DSU and the DebugBlock.

Some external interfaces, such as the main ACE or CHI master interface, support a clock enable input to allow the external logic to run at a lower, synchronous, frequency.

While there is no functional requirement for the clocks to have any relationship with each other, the DSU is designed with the following expectations to achieve acceptable performance:

- CORECLK[CN:0] is dynamically scaled to match the performance requirements of each core.
- SCLK frequency affects the L3 hit latency and, therefore, it is important for achieving good performance. For best performance, Arm recommends running SCLK as close to CORECLK[CN:0] frequency as possible. However to reduce dynamic and leakage power, targeting a lower frequency might be required. Running SCLK at least approximately 75% of the CORECLK[CN:0] frequency might give an appropriate balance for many systems.
- SCLK can run at synchronous 1:1 or 2:1 frequencies with the external interconnect, avoiding the need for an asynchronous bridge between them.
- PCLK, ATCLK, and GICCLK can run at the same frequency as the relevant SoC components that they connect to. This would typically be approximately 25% of the maximum CORECLK[CN:0] frequency.
- PERIPHCLK contains the architectural timers, and software performance can be impacted if reads to these registers take too long. Therefore, Arm recommends that PERIPHCLK is run at least 25% of the maximum CORECLK[CN:0] frequency.

4.2 Resets

The DSU requires reset signals for each of the cores, internal logic, and external interfaces.

Table 4-2: DSU reset signals

Signal	Description
nCPUPORESET[CN:0]	The per-core primary Cold reset signal for all resettable registers in the CORECLK domain including debug registers, ETM registers, and RAS registers.
nCORERESET[CN:0]	The per-core Warm reset signal for all resettable registers in the CORECLK domain excluding the debug registers, ETM registers, and RAS registers.
nPRESET	A single cluster-wide reset signal for all resettable registers in the PCLK domain.
nSPORESET	A single cluster-wide Cold reset signal for all resettable registers in the SCLK domain.
nSRESET	A Warm reset signal for all resettable registers in the SCLK domain excluding RAS registers.
nATRESET	A single cluster-wide reset signal for all resettable registers in the ATCLK domain.
nGICRESET	A single cluster-wide reset signal for all resettable registers in the GICCLK domain.
nPERIPHRESET	A single cluster-wide reset signal for all resettable registers in the PERIPHCLK domain.
nMBISTRESET	A single cluster-wide reset signal that acts on all resettable registers in the CORECLK and SCLK domains.

The nMBISTRESET is active low and must be driven high for functional mode. A single top level nMBISTRESET is utilized to reset the IP in preparation for MBIST operations.

All reset inputs can be asserted (HIGH to LOW) and deasserted (LOW to HIGH) asynchronously. Reset synchronization logic inside the DSU ensures that reset deassertion is synchronous for all resettable registers inside those reset domains. The core clock does not need to be present for reset assertion, but it must be present for reset deassertion to ensure reset synchronization.

AArch32 (EL3) to AArch64 Execution state transition at reset

At *Exception level 3* (EL3), cores can only transition between AArch32 and AArch64 states at reset. The Execution state after reset is controlled by the AA64nAA32[PE:0] configuration signals. These signals are only sampled at reset.

To reset a core and change Execution state from software, a Warm reset request can be made by setting the RR bit of the RMR system register (from AArch32) or the RMR_EL3 register (from AArch64). Following the register write and executing a `WFI` instruction, the cluster automatically resets the core without requiring any action by the external reset controller. The hardware automatically cleans and invalidates all the caches and safely disconnects the core from cluster before the reset is asserted.

Note

Your implementation might include cores that do not support execution in AArch32 state at EL3.

5. Power management

This chapter describes the power domains and the power modes in the DSU.

5.1 About DSU power management

The DSU supports a range of low-power modes and cache RAM powerdown modes.

The DSU supports the following power modes:

On mode

On mode is the normal mode of operation where all the core and DSU functionality is available. The DSU individually disables internal clocks, and inputs to unused functional blocks. Only the logic that is in use consumes dynamic power.

Functional retention mode

Functional retention mode allows the L3 cache and snoop filter RAMs to be put temporarily in to a retention state while the L3 cache is not being accessed. The contents of the cache RAMs are retained.

Memory retention mode

Memory retention mode allows the L3 cache RAMs to be held in retention while the rest of the cluster is powered down. Keeping the RAMs in retention reduces the energy cost of writing dirty lines back to memory and reduces the cluster response time on powerup. It is not possible to snoop the cache in this mode. Therefore it is important that no other external coherent agents are active (for example, cores external to the cluster, or other coherent devices). In practice, this mode can only be used in a coherent system when the cluster is the only active agent.

Off mode

In Off mode, power is removed completely, and no state is retained. To avoid losing data, the DSU cleans and invalidates the L3 cache before taking the cluster out of coherence.

The DSU supports clock, voltage, and power domains that can be controlled by external logic.

The cluster, along with power management software, gives operating requirement hints to an external power controller. The power controller is responsible for:

- Coordinating power management with the rest of the SoC;
- Switching and isolating power and voltage domains; and
- Controlling clock gating cells.

5.2 Power mode control

Power management control is distributed between power management software, the cluster, and an external power controller.

The cluster supports a set of power operating requirements which act as hints to the external power controller. The operating requirements indicate: the required cache capacity, the RAM retention mode, and whether the cluster logic can be powered up or down. The power controller controls the logic clamps and power switches required to put the RAMs and logic into low-power operation.

Software sets the operating requirements by writing to the following system registers:

***Cluster Power Control Register* (CLUSTERPWRCTLR_EL1)**

To request partial L3 cache powerup or powerdown, and to enable RAM retention capabilities.

***Cluster Powerdown Register* (CLUSTERPWRDN_EL1)**

To request the power mode that the cluster is to enter, after all cores have powered off. For example, memory retention mode.

The operating requirements are signaled to the power controller through the cluster P-Channel interface. The power controller responds to a change of operating requirements by sequencing the transition between lower or higher power modes.

Power management algorithms can use *Cluster L3 Hit Counter Register* (CLUSTERL3HIT_EL1) and *Cluster L3 Miss Counter Register* (CLUSTERL3MISS_EL1) system registers to determine when to powerup or powerdown cache portions.

The status of the power settings is indicated in the *Cluster Power Status Register* (CLUSTERPWRSTAT_EL1) system register.

The cluster receives power mode transition requests from the power controller and checks the validity of each transition. If the transition is supported, the cluster accepts the request. If the transition is not supported, the cluster denies the request. If the cluster accepts the request, the power controller can switch power domains off as appropriate.

The cluster automatically performs any internal operations required by a mode transition, before accepting the new mode. The internal actions performed by the cluster include: gating clocks, flushing caches, and disabling coherency.

Note

If L3 RAM retention is not implemented, CLUSTERPWRCTLR_EL1.L3 data RAM retention control must be left in the reset state.

5.3 Communication with the power controller

The cluster provides a P-Channel interface to allow the external power controller to set the operating power mode of the cluster, in response to requests from the cluster.

The cluster indicates the operating requirements on the CLUSTERPACTIVE bus. The power controller can then request a new power mode. The power controller indicates the requested mode on the CLUSTERPSTATE bus and asserts the CLUSTERPREQ handshake signal.

When the cluster has performed all the actions that are required in preparation for the power mode transition, the cluster accepts the request by asserting CLUSTERPACCEPT. If the request is not valid, the cluster denies the request by asserting CLUSTERPDENY. Reasons for denying the request might be, requesting an incorrect mode transition, or the requested mode is no longer appropriate because the L3 operating mode has changed.

At reset, the cluster reads the initial mode set by the power controller on the CLUSTERPSTATE bus.

Note

Arm recommends that CLUSTERPREQ is asserted at reset. If CLUSTERPREQ is not asserted at reset, the power controller must wait for 40 PERIPHCLK cycles after reset is deasserted before it drives CLUSTERPREQ HIGH.

5.4 L3 RAM power control

Power savings can be made by putting parts, or all, of the L3 control logic and L3 cache RAM into retention modes, or powering down portions of the RAM.

5.4.1 L3 cache partial powerdown

Sections of the L3 cache that are known as portions, can be independently powered down to reduce RAM leakage power.

When all the L3 cache capacity is not required, then the L3 cache size can be reduced by powering down one or more of these portions. For instance, for small memory footprint workloads.

The L3 cache RAMs are organized to allow separate control of groups of cache ways. Each group has four cache ways. The L3 data RAMs are organized into two equal sized portions, where each portion consists of two groups of cache ways. The L3 tag RAMs are organized into four equal sized portions, with each portion corresponding to each group of cache ways.

Power control can be applied independently to each portion.

The operating cache capacity can be selected from: all, ¾, ½, ¼, and none.

The following table shows how the available operating cache capacities relate to the RAM power enables for each portion.

Table 5-1: L3 Cache capacity and RAM power enables

Cache capacity	Tag RAM				Data RAM	
	0	1	2	3	0	1
None	Off				Off	
¼	On	Off			On	Off
½	On		Off		On	Off
¾	On			Off	On	
All on	On				On	

This table shows that one portion of the tag RAM for the ¼ and ¾ cache capacities are always powered down. Therefore, these cache capacities may not achieve the same power savings of the ½ and all L3 capacities, respectively.

The external power controller requests the required L3 cache capacity, through the cluster P-Channel, in response to power management software setting the operating requirements.

Note

Confirm with your implementer whether RAM powerdown is supported. If RAM powerdown is supported, then ensure that the necessary logic clamps and power switches are implemented.

Memory transactions from the cores can still be processed when all of the portions are off, and while a portion is being powered on or off.

For more information on L3 cache slices and portions, see 6.7 Cache slices and portions on page 62.

5.4.2 L3 RAM retention

The *DynamIQ™ Shared Unit* (DSU) supports the Functional retention and Memory retention power modes.

Functional retention

Functional retention mode (SFONLY FUNC_RET, ¼FUNC_RET, ½FUNC_RET, ¾FUNC_RET, or FULL FUNC_RET mode) allows the L3 cache and snoop filter RAMs to be put temporarily into retention while the L3 cache is not being accessed.

When the L3 cache has not been accessed for a period of time, the DSU signals to the power controller that the L3 cache RAM can be put into retention. While in functional retention mode, core and snoop requests can still be received. On receiving a request, the DSU signals to the power controller to take the RAMs out of retention. The core or snoop request is stalled until the power controller signals to the DSU to enter an ON mode

(SFONLY ON, ¼ON, ½ON, ¾ON, or FULL ON). When the request is completed, the DSU signals to the power controller that the RAMs can be put back into retention.

The *Cluster Power Control Register* (CLUSTERPWRCTLR) system register determines the duration of inactivity before the DSU requests the RAMs to be put into retention.

Memory retention

Memory retention mode (¼MEM_RET, ½MEM_RET, ¾MEM_RET, or FULL_MEM_RET mode) allows the L3 cache RAMs to be put into retention. Memory retention mode also disables the control logic in the DSU, and allows the snoop filter and *Long-Term Data Buffer* (LTDB) RAMs to be powered off.

Memory retention mode can be entered when the DSU is idle and all cores are OFF. In this mode, the L3 cache cannot process ACP or snoop requests.

Software can request the DSU to enter memory retention by setting the Memory retention required bit in CLUSTERPWRDN_EL1 before that core is powered OFF. After all cores have transitioned to the OFF power mode, and the DSU becomes idle, the DSU indicates MEM_RET on its P-Channel CLUSTERPACTIVE signal.

Even if CLUSTERPACTIVE indicates MEM_RET, the DSU can still accept a P-Channel request to transition to the OFF mode.

5.5 Power modes

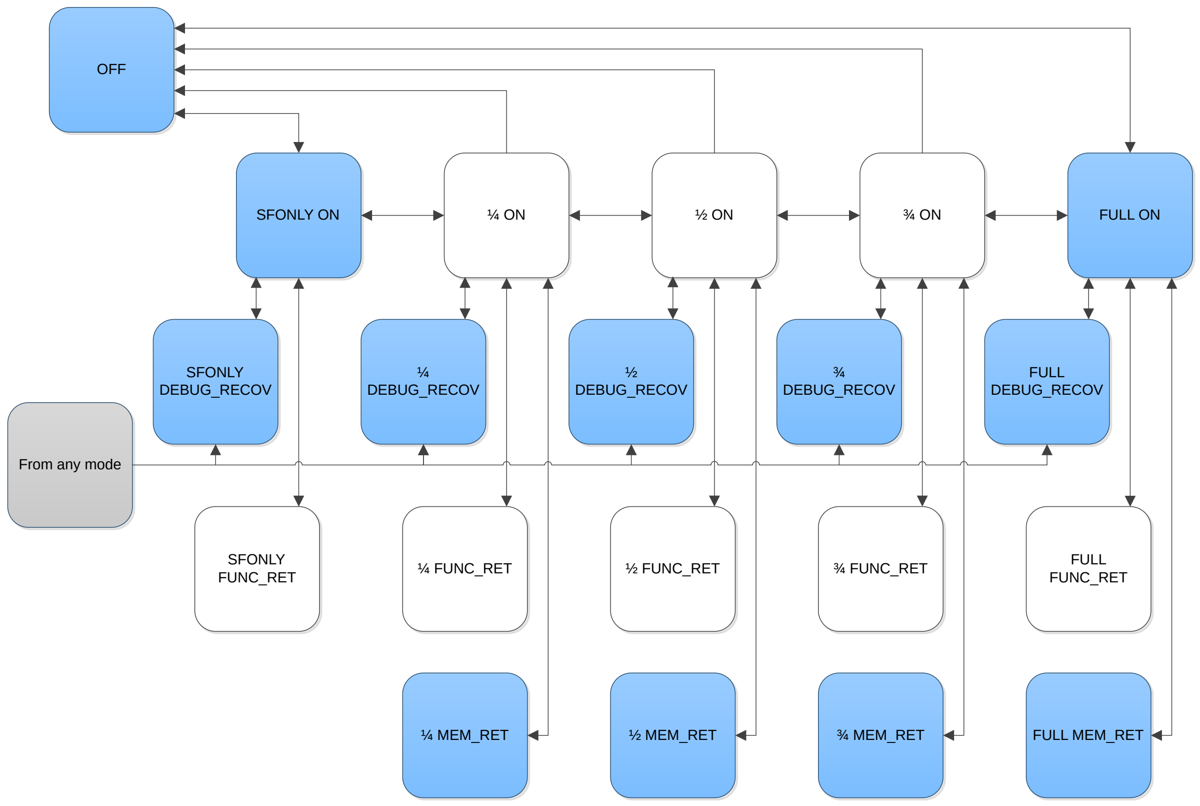
The external power controller can request a new operating power mode in response to the operating requirements indicated by the DSU.

5.5.1 Power mode transitions

The *DynamIQ™ Shared Unit* (DSU) supports a set of power modes. The power controller can request the DSU to transition between modes.

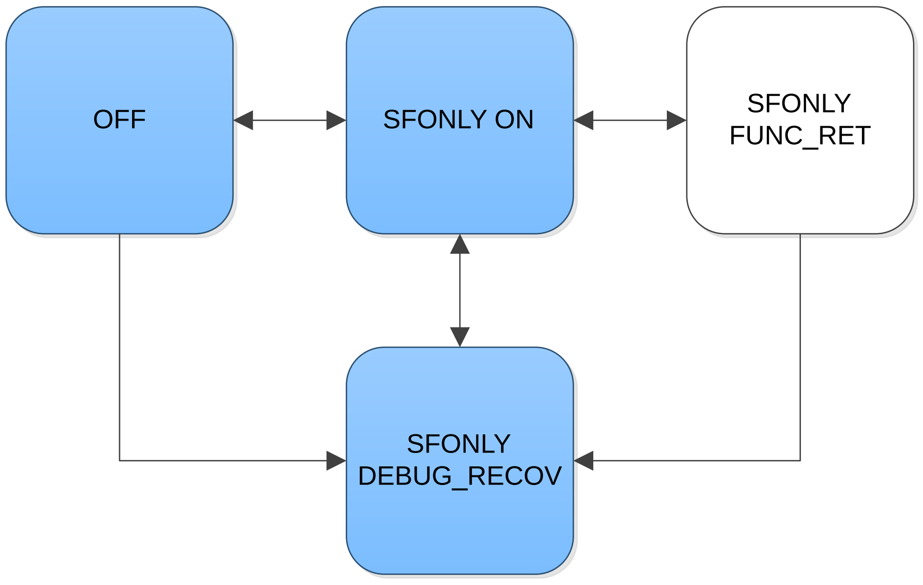
The following diagram shows the supported DSU power modes, and the permitted transitions between them. The blue modes indicate the modes that the DSU can be initialized to at reset. See 5.3 Communication with the power controller on page 38

Figure 5-1: DSU power mode transitions



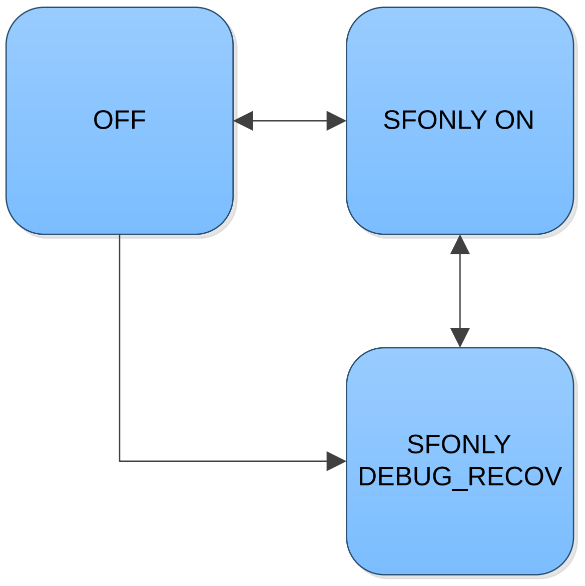
The following diagram shows the supported DSU power modes when L3 is not implemented.

Figure 5-2: DSU power mode transitions, no L3



The following diagram shows the supported DSU power modes when Direct connect is implemented.

Figure 5-3: DSU power mode transitions, Direct connect



Note

In a configuration that implements Direct connect, there is no snoop filter present. The power modes are named SFONLY for consistency with other configurations.

FULL ON

In this mode, all DSU logic, snoop filter, and L3 cache RAMs are powered up and fully operational.

The DSU can be initialized into the FULL_ON mode. In this case, it is treated as an implicit transition from the OFF mode so the L3 cache and snoop filter are invalidated. When a transition to the FULL ON mode is completed, the cache and snoop filter are accessible and coherent. The L3 cache does not require any configuration from software.

SFONLY ON, ¼ON, ½ON, ¾ON

In these modes, the DSU logic and snoop filter RAMs are powered up but some of the L3 cache RAMs remain powered down.

The DSU can be initialized into the SFONLY ON mode. In this case, it is treated as an implicit transition from the OFF mode. When a transition to the SFONLY ON mode is completed, the snoop filter is accessible and coherent without needing any configuration from software.

Note

Transitions between the ON modes are only allowed in incremental steps. For example, a transition directly from ½ON to FULL ON is not permitted. The CLUSTERPACTIVE outputs reflect this, so that CLUSTERPACTIVE does not indicate a required mode that cannot be directly reached from the current mode.

SFONLY FUNC_RET, ¼FUNC_RET, ½FUNC_RET, ¾FUNC_RET, FULL FUNC_RET

In these modes, the DSU is powered up, the snoop filter and L3 cache RAMs are in retention, meaning that the RAMs are inoperable but with state retained.

If a request from a core, or a snoop from the system, is required to be serviced:

- The DSU indicates that a transition to ON is required using CLUSTERPACTIVE.
- The request is stalled until the DSU enters one of the ON modes.

When the RAMs are in retention, the clock to the RAMs is automatically gated outside of the retained domain.

¼MEM_RET, ½MEM_RET, ¾MEM_RET, FULL MEM_RET

In these modes, the DSU logic is powered down. The L3 cache RAMs are in retention and the snoop filter RAMs are powered down.

All cores in the cluster must be in the OFF mode to allow entry into any one of these modes.

Because the cache contains data that is accessible from the rest of the system, these modes must only be used if there are no other coherent agents in the system that are active and might need to access the data.

OFF

In this mode, all DSU logic, snoop filter, and L3 cache RAMs are powered down.

The DSU can be initialized to this mode on Cold reset.

Debug recovery mode

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

By default, the DSU invalidates the cache and snoop filter when there is a transition from OFF to an ON mode. In Debug recovery mode, cache invalidation is disabled allowing the contents of the L3 cache that were present before the reset to be observable after the reset. The contents of the L3 cache and snoop filter are preserved and are not altered on the transition back to the ON mode.

To enter Debug recovery mode, the P-Channel is initialized to DEBUG_RECOV, and the DSU is cycled through a reset.

To preserve the RAS state and cache contents, a transition to the Debug recovery mode can be 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.

Note

- Debug recovery mode is strictly for debug purposes. It must not be used for functional purposes, because correct operation of the cluster is not guaranteed when entering this mode.
- Debug recovery mode can occur at any time with no guarantee of the state of the cluster. 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 transactions might complete after the reset when the cluster 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 might get a response and disturb the contents of the caches, or it might not get a response and cause a system deadlock.
- If the cluster is already in the middle of a power mode transition on the P-Channel, a clock gating transition on the SCLK Q-Channel, or the cluster is in Warm reset, then it might not be possible to enter DEBUG_RECOV without a Cold reset of the cluster.
- The SFONLY, ¼, ½, ¾ and FULL DEBUG_RECOV modes are all functionally equivalent. You can transition to any of these modes from any other mode, and to transition to any ON mode from any DEBUG_RECOV mode. However, you

must choose the correct ON mode corresponding to the L3 cache portions that were in use before Debug recovery mode.

5.5.2 Power mode transition behavior

When the power controller requests a transition between power modes, the *DynamIQ™ Shared Unit* (DSU) automatically performs some actions before accepting the new mode.

Your SoC might implement additional actions. When transitioning from a lower power to higher power mode, these additional actions are performed before the power controller requests a new mode. When transitioning from a higher power to lower power mode, these actions are performed after the DSU accepts the new mode.

The following table shows the permitted mode transitions and behavior that is associated with each transition.

Table 5-2: Power mode transition behavior

Start mode	End mode	DSU behavior	Partner implemented behavior
OFF	ON	The L3 cache and snoop filter are initialized. The cluster is brought into coherency with the rest of the system.	Power applied, isolation disabled.
MEM_RET	ON	The cluster is brought into coherency with the rest of the system. The snoop filter RAMs are initialized.	Power applied, isolation disabled.
ON	FUNC_RET	Waits for all memory transactions to complete. The clock to the retention domain is gated. L3 cache and snoop filter RAMs are put into retention.	RAM clamps and isolation enabled.
FUNC_RET	ON	L3 cache and snoop filter RAMs are taken out of retention.	RAM clamps and isolation disabled.
FULL ON	¾ON	Decreasing available cache ways. Relevant ways in L3 cache are cleaned and invalidated.	RAM clamps and isolation are enabled for relevant ways.
¾ON	½ON		
½ON	¼ON		
¼ON	SFONLY ON		
SFONLY ON	¼ON	Increasing available cache ways. Relevant ways in L3 cache are initialized.	RAM clamps and isolation are disabled for relevant ways.
¼ON	½ON		
½ON	¾ON		
¾ON	FULL ON		
ON	OFF	Waits for all memory transactions to complete. L3 cache allocation is disabled. The L3 cache is cleaned and invalidated. The cluster is removed from system coherency.	DSU clamps and isolation to the rest of the system are enabled.
ON	MEM_RET	Waits for all memory transactions to complete. The cluster is removed from system coherency. This mode is only useful when the cluster is the only master active.	DSU clamps and isolation to the rest of the system are enabled.
Any	DEBUG_RECOV	-	-
DEBUG_RECOV	ON	The cluster is brought into coherency with the rest of the system.	Reset is applied, clamps and isolation are disabled.

5.5.3 Interlocks between core and DSU P-Channels

To ensure that correct operation is maintained, and to allow a cluster powerdown to be abandoned, there are interlocks between the core and DSU P-Channels:

- If a core P-Channel request is made while the DSU is not in an ON or FUNC_RET mode, the core request stalls until the DSU has reached the appropriate mode.

Note

If the DSU is in the MEM_RET or OFF mode, the power controller must transition the DSU to the ON mode to avoid deadlock when signaling a core powerup request.

- If the DSU is requested to go to MEM_RET or OFF while not all cores are OFF, the request is denied.
- If the DSU is in the process of transitioning from ON to OFF (particularly when flushing the L3 cache which can take a long time) and a core is requested to leave the OFF mode, the L3 cache flush is abandoned and the DSU P-Channel request is denied.

5.5.4 Power mode encoding

The power mode is encoded on the CLUSTERPSTATE P-Channel bus.

The following table shows the encoding of the power mode.

Table 5-3: Cluster power domain CLUSTERPSTATE values

Power mode	CLUSTERPSTATE Value	Logic	Snoop filter and LTDB RAMs	L3 Tag ways 0-3, L3 Data portion 0, and L3 victim RAMs	L3 Tag ways 4-7 RAMs	L3 Tag ways 8-11 and L3 Data portion 1 RAMs	L3 Tag ways 12-15 RAMs
FULL_ON	`0b1001000`	Powered up	Powered up	Powered up	Powered up	Powered up	Powered up
¾ ON	`0b0111000`						Powered down
½ ON	`0b0101000`					Powered down	
¼ ON	`0b0011000`				Powered down		
SFONLY ON	`0b0001000`			Powered down			
FULL_FUNC_RET	`0b1000111`		Retention	Retention	Retention	Retention	Retention
¾ FUNC_RET	`0b0110111`						Powered down
½ FUNC_RET	`0b0100111`					Powered down	
¼ FUNC_RET	`0b0010111`				Powered down		
SFONLY FUNC_RET	`0b0000111`			Powered down			

Power mode	CLUSTERPSTATE Value	Logic	Snoop filter and LTDB RAMs	L3 Tag ways 0-3, L3 Data portion 0, and L3 victim RAMs	L3 Tag ways 4-7 RAMs	L3 Tag ways 8-11 and L3 Data portion 1 RAMs	L3 Tag ways 12-15 RAMs
FULL_MEM_RET	`0b1000010`	Powered down	Powered down	Retention	Retention	Retention	Retention
¾ MEM_RET	`0b0110010`						Powered down
½ MEM_RET	`0b0100010`					Powered down	
¼ MEM_RET	`0b0010010`				Powered down		
OFF	`0b0000000`			Powered down			
FULL DEBUG_RECOV	`0b1001010`	Powered up or down[1]	Powered up or retention[1]	Powered up or retention[1]	Powered up or retention[1]	Powered up or retention[1]	Powered up or retention[1]
¾ DEBUG_RECOV	`0b0111010`						Powered down[1]
½ DEBUG_RECOV	`0b0101010`					Powered down[1]	
¼ DEBUG_RECOV	`0b0011010`				Powered down[1]		
SFONLY DEBUG_RECOV	`0b0001010`			Powered down[1]			

5.6 Power operating requirements

The DSU power operating requirements are indicated to the power controller by asserting bits on the CLUSTERPACTIVE bus.

The meaning of each bit is described in the following table.

Table 5-4: L3 power domain CLUSTERPACTIVE bit positions

CLUSTERPACTIVE bit	Power operating requirement	Description
[19]	Cache ways 12-15	Indicates the required mode of the cache way RAMs. **0** Cache ways are not required and can be requested to power down. **1** Cache ways are required to be active.
[18]	Cache ways 8-11	
[17]	Cache ways 4-7	
[16]	Cache ways 0-3	
[8]	On	DSU logic and RAMs are required to be powered up.
[7]	Functional retention	DSU logic is active, RAMs can be put into retention.
[2]	Memory retention	DSU logic can be off, RAMs required to be retained.
[0]	Off	Always 0. Request to power down DSU logic and RAMs should be accepted when the entire CLUSTERPACTIVE bus is zero.

[1] In DEBUG_RECOV, the power mode does not need to accurately reflect the L3 portions that are powered up.

The following table shows how the CLUSTERPACTIVE bit combinations are mapped to power modes.

Table 5-5: L3 memory system power domain requested modes

CLUSTERPACTIVE bit								Requested mode
[19]	[18]	[17]	[16]	[8]	[7]	[2]	[0]	
Cache ways 12-15	Cache ways 8-11	Cache ways 4-7	Cache ways 0-3	On	Functional retention	Memory retention	Off	
1	X	X	X	1	X	X	X	FULL_ON
0	1	X	X	1	X	X	X	¾ ON
0	0	1	X	1	X	X	X	½ ON
0	0	0	1	1	X	X	X	¼ ON
0	0	0	0	1	X	X	X	SFONLY ON
1	X	X	X	0	1	X	X	FULL_FUNC_RET
0	1	X	X	0	1	X	X	¾ FUNC_RET
0	0	1	X	0	1	X	X	½ FUNC_RET
0	0	0	1	0	1	X	X	¼ FUNC_RET
0	0	0	0	0	1	X	X	SFONLY FUNC_RET
1	X	X	X	0	0	1	X	FULL_MEM_RET
0	1	X	X	0	0	1	X	¾ MEM_RET
0	0	1	X	0	0	1	X	½ MEM_RET
0	0	0	1	0	0	1	X	¼ MEM_RET
X	X	X	X	0	0	0	X	OFF

Note

The CLUSTERPACTIVE and COREPACTIVEx outputs are hints for the desired power mode. In some cases it might not be possible for the cluster to transition directly from the current mode to the requested mode. The power controller must be aware of the valid transitions and request transitions through intermediate modes if a direct transition is not valid. For example, if the cluster is in FUNC_RET and CLUSTERPACTIVE indicates MEM_RET, then the power controller must first request a transition from FUNC_RET to ON, before requesting a transition from ON to MEM_RET.

5.6.1 Power control for DFT

When DFT activities are being performed on the cluster, the P-Channel controls are not functional.

This means that if the cluster is in MBIST mode, or ATPG scan is in progress, then the CLUSTERPACTIVE or COREPACTIVEx outputs can take any value and the P-Channel protocol will not be followed. The SoC power control components must be aware of the DFT activities and must ensure that the logic and RAMs are appropriately powered without using the P-Channel.

5.7 Wait For Interrupt and Wait For Event

To reduce dynamic power, each core can request entry in to a low-power state using the *Wait For Interrupt* (WFI) and *Wait For Event* (WFE) instructions.

In the low-power state, most of the clocks in a core are disabled while keeping the core powered up. This state reduces the power that is drawn to the static leakage current, leaving a small clock power overhead to enable the core to wake up.

In addition to the per-core WFI and WFE low-power states, the clock to (almost all) the SCU and L3 logic is automatically disabled when the cluster is sufficiently idle.

A `WFI` or `WFE` instruction completes when:

- All outstanding load instructions are completed.
- All store instructions are completed.
- All cache and TLB maintenance operations are completed.
- All bus traffic to the L3 is completed.

While a core is in the low-power state, the clocks in the core are temporarily enabled under the following conditions:

- A snoop request from the L3 cache that the L1 data cache or L2 unified cache must service.
- A cache or TLB maintenance operation that the core L1 caches or TLB must service.
- An APB access to the debug or trace registers residing in the core power domain.
- An access request from the GIC distributor to the GIC CPU interface.

While the clocks in the core are temporarily enabled, the core remains in the WFI or WFE low-power state.

WFE wake up event signaling

- A *Send Event* (SEV) instruction signals a `WFE` wake up event to other clusters by asserting the EVENTOREQ output.
- The EVENTIREQ input indicates that another cluster or system component has signaled a `WFE` wake up event.

System global exclusive monitor signaling

Any global exclusive monitor in the system must generate an event when it is cleared. This event must be signaled to the cluster using the EVENTIREQ input.

5.8 Clock, voltage, and power domains

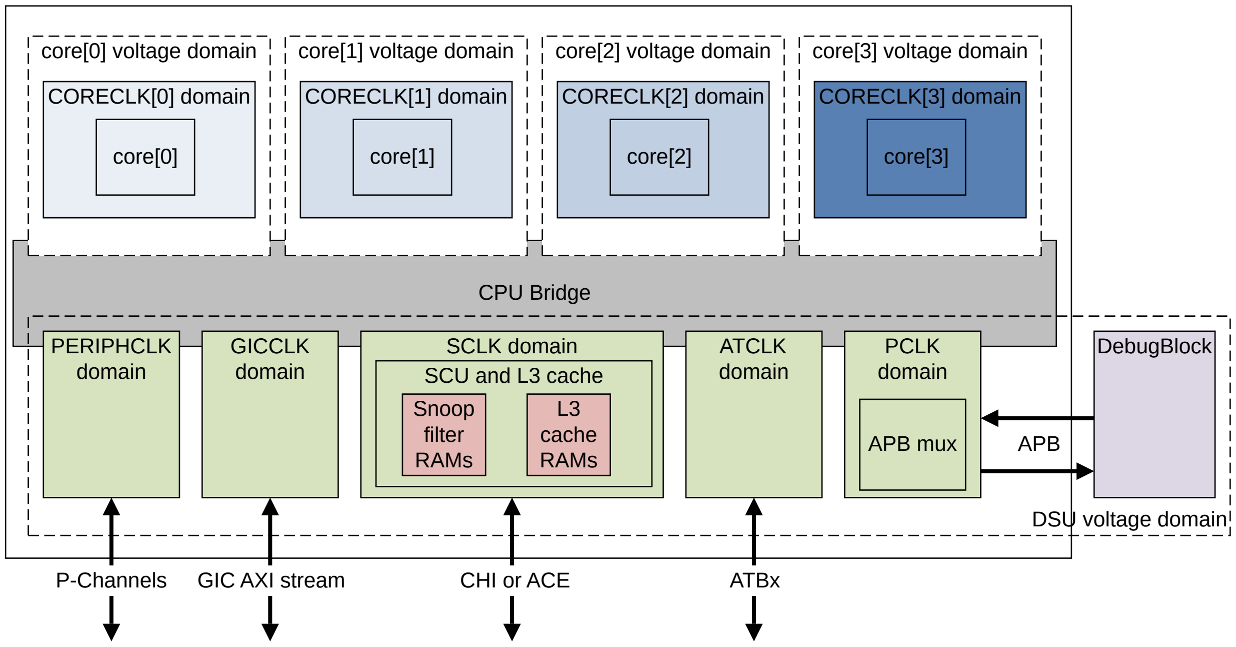
The DynamIQ™ cluster microarchitecture supports multiple clock, voltage, and power domains.

The number of domains that are implemented depends on the choices made by the SoC implementer. There might be fewer in your SoC.

The following diagram shows the clock, voltage, and power domains supported by the DSU and cores:

- Voltage domains are indicated by dashed outlines.
- Blocks that are in the same power domain have the same color.

Figure 5-4: DSU Clock, voltage, and power domains



Clock domains

Each core can be implemented in a separate clock domain. The DSU has multiple clock domains.

The CPU Bridge contains all asynchronous bridges for crossing clock domains, and is split with one half of each bridge in the core clock domain and the other half in the relevant cluster domain. Each core can be implemented with or without an asynchronous bridge. If the asynchronous bridge is not implemented, the core is in the SCLK clock domain.

The DebugBlock can be implemented in the PCLK domain. However, the DebugBlock might be implemented in a different domain. In this case, asynchronous bridges must be implemented on the APB interfaces between the DebugBlock and the cluster.

Each clock domain, except the PERIPHCLK domain and CORECLK domains, has an associated Q-Channel, which allows the DSU to request the external clock controller to gate the clock on or off.

Voltage domains

Each core can be implemented in a separate voltage domain. The DSU has a single separate voltage domain. This allows, for example, the DSU to be in the same voltage domain as the SoC interconnect and other system components.

Power domains

Each core can be implemented in one, or more, separate power domains.

Additional power domains that can be implemented include:

- L3 cache RAM portions.
- SCU.
- DebugBlock.

5.9 Cluster powerdown

The cluster is taken out of coherence with the interconnect automatically when it is powered down. No software sequence is required.

After receiving the request to enter power off mode from the power controller, the DSU cleans and invalidates the L3 cache, and communicates with the interconnect to disable snoops into the cluster. All cores must be in the OFF mode before the cluster is powered down.

Note

To support automated removal from coherency, the interconnect must support the SYSCOREQ and SYSCOACK handshake protocol. If not, the SoC is responsible for programming the interconnect to remove the cluster from coherency.

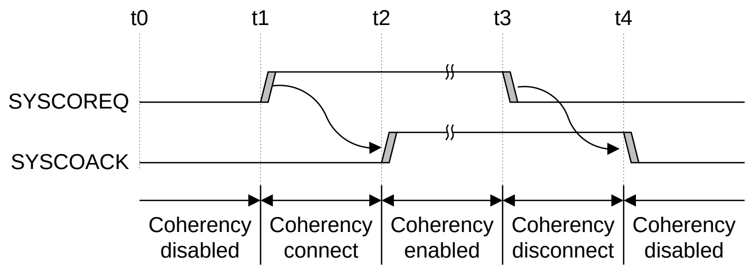
5.9.1 Transitioning in and out of coherency

The *DynamIQ™ Shared Unit* (DSU) provides a hardware mechanism for taking the cluster in and out of coherence with the system interconnect. The cluster enables coherency during powerup and disables it during powerdown.

The system interconnect can use the SYSCOREQ and SYSCOACK signals, to take the cluster in and out of coherence. If the system interconnect supports these signals, they can be connected directly. If the system interconnect does not support these signals, Arm recommends that they are connected to the power controller. In this case, the power controller must take any actions necessary to make the transition.

The following diagram shows the timing of SYSCOREQ and SYSCOACK.

Figure 5-5: SYSCOREQ SYSCOACK four-phase coherency handshake



To enable coherency, the cluster always asserts SYSCOREQ during powerup. When the system interconnect has enabled coherency, it asserts SYSCOACK and can then start sending snoop requests to the cluster. The cluster accepts snoop requests whenever either signal is asserted.

The cluster disables coherency during powerdown. The cluster deasserts SYSCOREQ and waits for the system interconnect to deassert SYSCOACK. The system interconnect must not deassert SYSCOACK until it can guarantee that: there are no further snoop requests to be sent, and that all snoop requests it has already sent have fully completed.

The signals must obey the following four-phase handshake rules:

- SYSCOREQ can only change when SYSCOACK is at the same level.
- SYSCOACK can only change when SYSCOREQ is at the opposite level.

5.9.1.1 Coherency signals naming convention

The *DynamIQ™ Shared Unit* (DSU) supports up to two CHI or ACE interfaces. Each set of coherency signals has a unique suffix.

The coherency signals are named as follows:

CHI master 0
SYSCOREQ and SYSCOACK.

CHI master 1
SYSCOREQM1 and SYSCOACKM1.

ACE master 0
SYSCOREQM0 and SYSCOACKM0.

ACE master 1
SYSCOREQM1 and SYSCOACKM1.

6. L3 cache

This chapter describes the optional L3 cache.

6.1 About the L3 cache

All the cores in the cluster share the L3 cache.

The shared L3 cache of the DSU provides the following functionality:

- A dynamically optimized cache allocation policy.
- Groups of cache ways can be partitioned and assigned to individual processes[2]. Cache partitioning ensures that processes do not dominate the use of the cache to disadvantage other processes.
- Support for stashing requests from the ACP and CHI interfaces. These stashing requests can also target any of the L2 caches of cores within the cluster.
- *Error Correcting Code* (ECC) protection is supported on the L3 cache data and tag RAMs.
- The cache can be implemented with either one or two cache slices. Each cache slice consists of data, tag, victim, and snoop filter RAMs and associated logic.

Note

- On powerdown, the DSU automatically performs cache cleaning, eliminating the need for software-controlled cache cleaning.
- The ACE master interface does not support cache stashing.

6.2 L3 cache allocation policy

The L3 cache data allocation policy changes depending on the pattern of data usage.

Exclusive allocation is used when data is allocated in only one core. Inclusive allocation is used when data is shared between cores.

For example, an initial request from core 0 allocates data in the L1 or L2 caches but not in the L3 cache. When data is evicted from core 0, the evicted data is allocated in the L3 cache. The allocation policy of this cache line is still exclusive. If core 0 refetches the line, it is allocated in the L1 or L2 caches of core 0 and removed from the L3 cache. The allocation policy of this cache line is still exclusive. If core 1 accesses this line for reading, it remains allocated in core 0 and is also allocated in both the core 1 and L3 caches. In this case, this line has an inclusive allocation, because it is being shared between cores.

[2] A process is an instance of a computer program.

6.3 L3 cache partitioning

The L3 cache supports a partitioning scheme that alters the victim selection policy to prevent a process from utilizing the entire L3 cache to the disadvantage of other processes.

Cache partitioning is intended for specialized software where there are distinct classes of processes running with different cache accessing patterns. For example, two processes A and B run on separate cores in the same cluster and therefore share the L3 cache. If process A is more data-intensive than process B, then process A can cause all the cache lines that process B allocates to be evicted. Evicting these allocated cache lines can reduce the performance of process B.

L3 cache partitioning is achieved by partition scheme IDs and groups of cache ways, where:

- Each group contains four ways.
- Each group can either be assigned as private to one or more partition scheme IDs, or be left unassigned.
- Each unassigned group can be shared between all eight partition scheme IDs.

Each core in the cluster must be assigned to at least one of the eight partition scheme IDs. L3 cache accesses from a given core can allocate into:

- Any cache way that belongs to a group that is assigned as private to the partition scheme ID of this core.
- Any cache way that belongs to an unassigned group that is shared by the entire cluster.

Up to four private L3 cache partitions can be created.

Note

If some cache ways are powered down, the number of ways in each L3 cache partition are reduced. This reduction in cache ways can result in insufficient ways being made available to cores, which may degrade their performance. Therefore Arm recommends that caution is used when powering down cache ways while using cache partitioning.

Partitioning setup example

The following example illustrates how the control registers can be programmed to partition the L3 cache into these three separate partitions:

- One partition that the hypervisor owns and uses for ACP and stashes.
- Two partitions that the OS assigns to processes running on the cores.

1. Software, running at EL3, sets ACTLR_EL3[10] to `0b1` and ACTLR_EL3[11] to `0b1` to delegate control of the partitioning to EL2.
2. The hypervisor, running at EL2, sets the CLUSTERPARTCR_EL1 to `0x00008601` to configure these three L3 cache partitions:
    - Scheme ID 0 (1/4 of the cache).
    - Scheme ID 2 (1/2 of the cache).

- Scheme ID 3 (1/4 of the cache).

3. The hypervisor sets the CLUSTERACPSID_EL1 to `0x0` and the CLUSTERSTASHSID_EL1 to `0x0`, to make ACP requests and stashes use partition scheme ID 0.
4. The hypervisor sets the CLUSTERTHREADSIDOVR_EL1 to `0x00060002`. This setting indicates that:
   - The upper two bits of the scheme ID are under the control of the hypervisor.
   - The lowest bit of the scheme ID can be controlled by the OS.
5. The hypervisor sets ACTLR_EL2[10] to `0b1` to delegate control of the CLUSTERTHREADSID_EL1 register to EL1.
6. The OS, running at EL1, can set the CLUSTERTHREADSID_EL1 to `0x0` or `0x1` to select between the two L3 cache partitions allocated to it. These would be mapped to scheme IDs 2 and 3 by the CLUSTERTHREADSIDOVR_EL1 register. The OS can update this register on context switches to select which partition each process has access to.

6.4 Cache stashing

Cache stashing allows an external agent to request that a line is brought (or stashed) into a cache.

Cache stashing can either be performed over the ACP interface, or the CHI master interface. Stash requests can target the L3 cache, or any of the L2 caches of cores within the cluster. The available stashing bandwidth is likely to be higher when stashing to the L3 cache.

On the CHI interface, stash requests (snoops) into both the L2 and L3 caches are supported by default. The field, StashLPIDValid, indicates the target of the stash, as follows:

- If the field is clear, then the stash is directed to the L3 cache.
- If this field is set, then the stash is directed to an L2 cache and the StashLPID field indicates which core is targeted.

On the ACP, by default, accesses are implicit stash requests into the L3 cache. Signal AWSTASHLPIDENS indicates a stash into L2. In this case, signal AWSTASHLPIDS[3:1] indicates which core to target, and AWSTASHLPIDS[0] indicates the thread. The choice of thread has no effect on the stash operation.

On the ACE master interface, cache stashing is not supported.

The cluster always attempts to allocate a stash request, unless it is heavily utilized and does not have any free buffers. In this case, the cluster drops a stash request to avoid a potential system deadlock.

Note

If there is a stashing snoop targeting an L3 (rather than a core) and there is no L3 cache, then the DSU will not perform the data pull, so the DSU will not send a SnpResp*_Read response. In this case, the DSU can send different responses for the different stashing snoops as the follows:

- For SnpStashShared, SnpStashUnique or SnpMakeInvalidStash, the DSU returns SnpResp_I response.
- For SnpUniqueStash, the DSU returns SnpRespData_I or SnpResp_I.

6.5 L3 cache ECC and parity

Error Correcting Code (ECC) and parity protection is supported on L3 cache RAM instances.

The ECC algorithm that is used is *Single Error Correct, Double Error Detect* (SECDED). It allows detection and correction of any 1-bit error and detection of any 2-bit error in all protected RAMs.

L3 cache RAMs implement ECC, as follows:

- SECDED ECC on the L3 data RAMs, per 64-bit.
- SECDED ECC on the L3 tag RAMs, per entry.
- SECDED ECC on the snoop filter RAMs, per entry.

If an error has no functional effect and only results in a minor change in performance, then the affected RAM is not protected. For example, victim selection RAMs are not protected with ECC or parity.

Error correction

When a correctable error is detected in the L3 cache data RAMs, the data is corrected inline before returning to the requester.

When a correctable error is detected in the L3 cache tag RAMs or the snoop filter RAMs the following correction mechanism is used:

- The value is corrected and written back to the source address (Read-Correct-Write).
- The lookup is replayed.

The DSU includes hardware that provides limited support for hard error correction. A hard error is a physical error in the RAM that prevents the correct value from being written. Hard errors can be corrected, but might cause a degradation in performance when the locations with errors are accessed.

Uncorrectable errors and data poisoning

If an error is detected as having two bits in error in a RAM protected by ECC, then this error is not correctable. In this case, the behavior depends on the type of RAM, as follows:

Data RAM or Long-Term Data Buffer RAM

When an uncorrectable error is detected in an L3 data RAM or *Long-Term Data Buffer* (LTDB) RAM, the chunk of data with the error is marked as poisoned. This poison status is then transferred with the data and stored:

- In the cache, if the data is allocated back into a cache.

- In the LTDB RAM, if the data is moved there.

The poison status is stored for every 64 bits of data.
If the interconnect supports poisoning, then the poison status is transferred with the data when the line is evicted or snooped from the cluster. No abort is generated when a line is poisoned. The abort is deferred until a load or instruction fetch consumes the poisoned data.

If the interconnect does not support poisoning and a poisoned cache line is evicted or snooped from the cluster, then the DSU generates an interrupt, nERRIRQ, to notify software that data has potentially been lost.

Note

Software can indicate if the interconnect supports poisoning or not by setting the interconnect data poisoning support bit in the Extended Control Register of the cluster. For details, see 11.8 CLUSTERECTLR, Cluster Extended Control Register on page 105.

Tag RAM

When an uncorrectable error is detected in an L3 tag RAM, then either the address or coherency state of the line is unknown, so the data cannot be poisoned. In this case, the line is invalidated and the DSU generates an interrupt, nERRIRQ, to notify software that data has potentially been lost.

Snoop filter tag RAM

When an uncorrectable error is detected in a snoop filter tag RAM, either the address or coherency state of the line is unknown, so the data cannot be poisoned. In this case, the snoop filter entry is invalidated, but the line remains present in one or more of the cores. The DSU generates an interrupt, nERRIRQ, to notify software that data has potentially been lost.

Note

Arm recommends that a system reset is performed as soon as possible, in response to this interrupt. Because the core caches and the snoop filter are inconsistent after this error, which can lead to **UNPREDICTABLE** behavior. The effect of the error depends on the type of core, but it could result in further data corruption, or deadlocks, making it impossible to cleanly recover from such an error.

Error reporting

Any detected error is reported in the *Error Record Primary Status Register* (ERR<n>STATUS) and the *Error Record Miscellaneous Register 0* (ERR<n>MISC0). These detected errors include 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 and the OF (overflow) bit of ERR<n>STATUS is set.

The ERRSELR_EL1 register provides the following two error records:

- 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 is for the cluster, which is shared between all cores in the cluster, and records any error in the L3 and snoop filter RAMs.

If enabled in the ERR<n>CTLR register (by setting one or more of the UI, FI, or CFI bits), all the detected errors generate a fault handling interrupt on:

- The nFAULTIRQ[0] pin (for L3 and snoop filter errors)
- The nFAULTIRQ[n+1] pin (for core n L1 and L2 errors)

Errors that cannot be corrected, also cause an abort or error handling interrupt because they can result in data corruption. The interrupt alerts software to this error so that it can either attempt to recover or restart the system. Some errors can be deferred by poisoning the data. These errors do not cause an abort at the time of the error, but only when the erroneous data is consumed.

The following describes the different types of errors that can occur and their effects:

- Uncorrectable errors in the L3 data RAMs when read by a core can cause a precise or imprecise Data Abort or Prefetch Abort, depending on the implementation of the core.
- Uncorrectable errors in the L3 data RAMs in a line when this line is being evicted from a cache, cause the data to be poisoned. The eviction might be because of a natural eviction, a line fill 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.
- Uncorrectable errors in the L1 tag or dirty RAMs, or in the L2 tag RAMs, cause the nERRIRQ[n +1] pin to be asserted for core<n>.
- Uncorrectable errors in the L3 tag RAMs or SCU snoop filter RAMs cause the nERRIRQ[0] pin to be asserted.

Note

Arm recommends that the nERRIRQ pins are 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 can be cleared by writing to the ERR<n>STATUS registers.

When a dirty cache line with an error on the data RAMs is evicted from the cluster, the write on the master interface still takes place. However, if the error is uncorrectable then:

- On ACE, the uncorrected data is written.
- On CHI, the uncorrected data is written but the data poison field indicates that there is a data error.

When a snoop hits on a line with an uncorrectable data error, the data is returned (if the snoop requires the data) and:

- On ACE, the nERRIRQ[0] pin is asserted.

Note

The snoop response does not indicate the error.

- On CHI, the snoop response indicates that either the data is poisoned, when the interconnect supports poisoning, or that there is an error.

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 not known whether the cache line is valid.

If an ACP access reads a cache line with an uncorrectable error, then it returns an ACP response to indicate a slave error.

Sometimes an error can be counted multiple times. For example, multiple accesses might read the location with the error before the line is evicted.

Error injection

Error injection is used to test out the error detection reporting and recording structure by deliberately inserting errors into the error reporting logic.

The injected errors are only pseudo-errors. They cause an error report to be signaled but the error injection does not corrupt the target location. Therefore, an injected pseudo error does not cause any automatic error correction logic to be activated.

Error injection uses the error injection and reporting registers to insert errors. Pseudo-errors are injected using the ERR1PFGCTLR, Error Pseudo Fault Generation Control Register.

Corrected and deferred pseudo-errors are triggered with accesses to *Long-Term Data Buffer* (LTDB) RAMs. Uncontainable pseudo-errors are triggered with accesses to the level 3 tag RAM instances. If the cluster is configured without a level 3 cache it will not be possible to trigger pseudo-errors in the DSU cluster.

Related information

Control registers on page 95

6.6 L3 cache data RAM latency

The L3 data RAM interface can be implemented with a configurable latency on the input and output paths.

The following options are available:

- Either a 1-cycle (the default) or 2-cycles write latency on the input path to the L3 data RAMs.
- Either a 2-cycles (the default) or 3-cycles read latency on the output path from the L3 data RAMs.

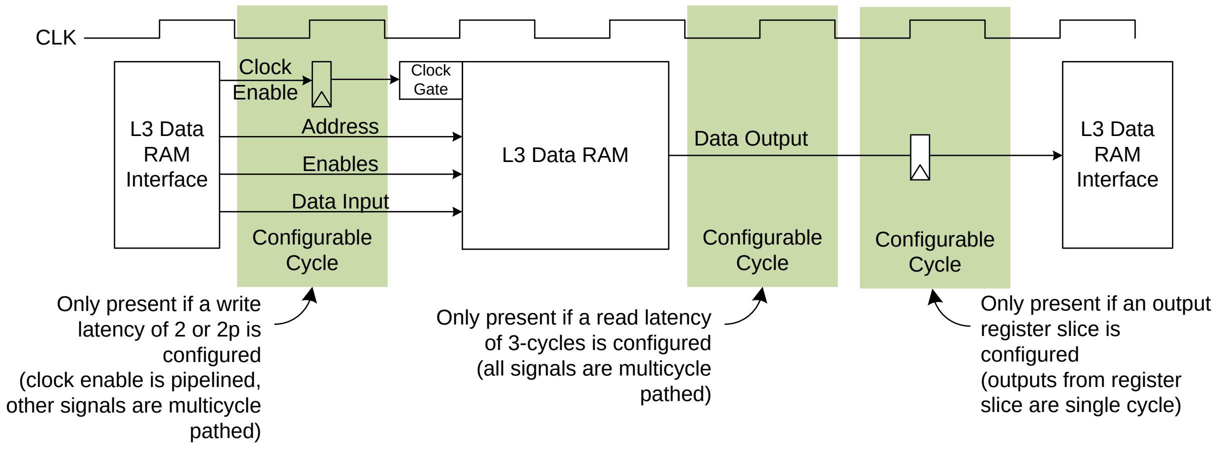
- An optional register slice on the output of the L3 data RAMs.

On the input paths, the 2p write latency keeps the RAM input signals stable for an extra cycle, which allows an extra cycle of hold timing on the RAM inputs. If a 2 or 2p write latency is requested on the input paths, then the RAM clock enable is pipelined and a multicycle path is applied to all other RAM input signals.

On the output paths, the 2-cycles read latency and 3-cycles read latency applies a multicycle path to all RAM output signals. The output of the optional register slice is single cycle and must never have a multicycle path applied.

The following diagram shows the L3 data RAM timing.

Figure 6-1: L3 cache data RAM latency


CLK
L3 Data RAM Interface
Clock Enable
Clock Gate
Address
Enables
Data Input
Configurable Cycle
L3 Data RAM
Data Output
Configurable Cycle
Configurable Cycle
L3 Data RAM Interface
Only present if a write latency of 2 or 2p is configured (clock enable is pipelined, other signals are multicycle pathed)
Only present if a read latency of 3-cycles is configured (all signals are multicycle pathed)
Only present if an output register slice is configured (outputs from register slice are single cycle)

An increase in RAM latency increases the L3 hit latency, which reduces performance.

Note

- Only use the 3-cycles read latency option if the RAM cannot meet the timing requirement of the 2-cycles latency. When the read latency is set to 3-cycles, the decrease in lookup throughput to one access every three core clock cycles is possibly more significant. Because when there are a series of L3 data RAM accesses close together, then the memory system could begin to back up.
- Use the register slice if the wire routing delay from the RAM to the SCU logic cannot meet the timing requirements.

The following table describes the impact on L3 data RAM performance with the different latency configuration parameters.

Table 6-1: L3 data RAM performance with different latency configurations

L3_DATA_WR_LATENCY	L3_DATA_RD_LATENCY	L3_DATA_RD_SLICE	L3 hit latency cycles	L3 lookup bandwidth	L3 write bandwidth
1	2	No	3	Access every 2-cycles	Access every 2-cycles

L3_DATA_WR_LATENCY	L3_DATA_RD_LATENCY	L3_DATA_RD_SLICE	L3 hit latency cycles	L3 lookup bandwidth	L3 write bandwidth
1	3	No	4	Access every 3-cycles	Access every 2-cycles
1	2	Yes	4	Access every 2-cycles	Access every 2-cycles
1	3	Yes	5	Access every 3-cycles	Access every 2-cycles
2	2	No	4	Access every 2-cycles	Access every 2-cycles
2	3	No	5	Access every 3-cycles	Access every 2-cycles
2	2	Yes	5	Access every 2-cycles	Access every 2-cycles
2	3	Yes	6	Access every 3-cycles	Access every 2-cycles
2p	3	No	5	Access every 3-cycles	Access every 3-cycles
2p	3	Yes	6	Access every 3-cycles	Access every 3-cycles

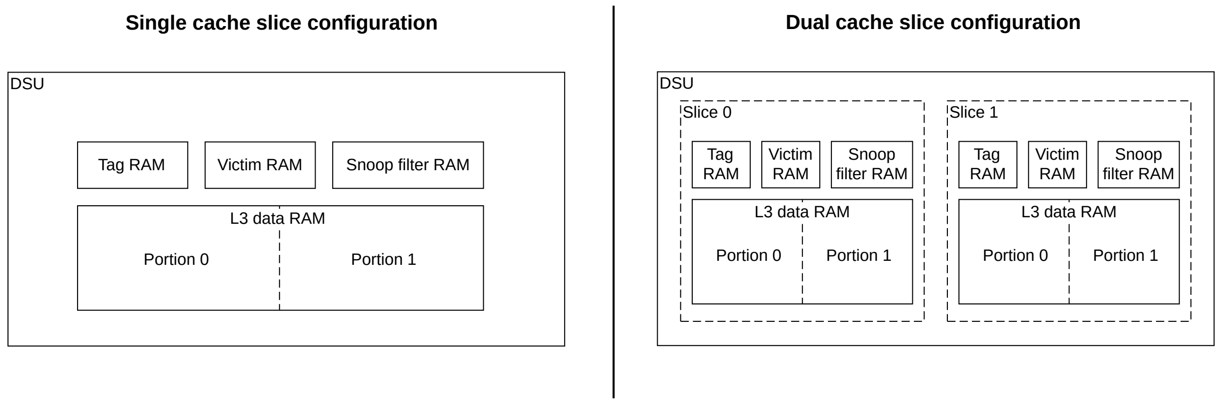
6.7 Cache slices and portions

The DSU is implemented as either having one or two L3 cache slices. A cache slice consists of data, tag, victim, and snoop filter RAMs and associated logic. A portion is a further subdivision of RAM in a cache slice.

When two L3 cache slices are implemented, the overall cache is effectively divided in two. Each cache slice has their own associated logic. But they are not independent because there is some shared logic between them. For each cache slice, the data RAM is subdivided into two portions, while the tag RAM is subdivided into four portions.

The following figure shows the differences between a single and a dual cache slice configuration.

Figure 6-2: Comparison between a single and dual L3 cache slice configuration



Dividing the L3 cache into two slices provides the following advantages:

- Improving the physical floorplan when implementing the macrocell, particularly for larger cache sizes.
- Increasing the bandwidth because the two slices can be accessed in parallel.

6.7.1 Cache slice and master port selection

For a dual cache slice implementation, requests are sent to a particular slice depending on the address and the memory attributes:

- For Normal Non-cacheable requests, the behavior depends on CLUSTERECTLR[0]. See 11.8 CLUSTERECTLR, Cluster Extended Control Register on page 105.
- For Cacheable requests, addresses are interleaved between slice 0 and slice 1, based on an address bit set by the `INTERLEAVE_ADDR_BIT` configuration parameter.
- Device requests are always sent to slice 0.

In a configuration with dual master ports, the slices directly correspond to the master ports, so an access sent to slice 0 uses master port 0.

See 7.1.1 Dual ACE interfaces on page 66 for more information.

6.7.2 Default number of cache slices

The configuration parameters that determine the default number of L3 cache slices.

Two cache slices are implemented, by default, when any of the following configuration options are chosen:

- More than four LITTLE cores are configured.
- One or more big cores are configured.

- The L3 cache size is greater than 1MB.
- A second ACE master port is configured.
- Either one or two 256-bit CHI master ports is configured.

A single slice is implemented by default for all other configurations.

Note

When there is either no L3 cache or an L3 cache that is 256KB, 512KB, or 1MB in size and there is only a single master port, then you can override the default number of L3 cache slices by using the `NUM_SLICES` configuration parameter. For more information, see 2.3 Implementation options on page 18.

6.7.3 Implementing a 1.5MB or 3MB L3 cache

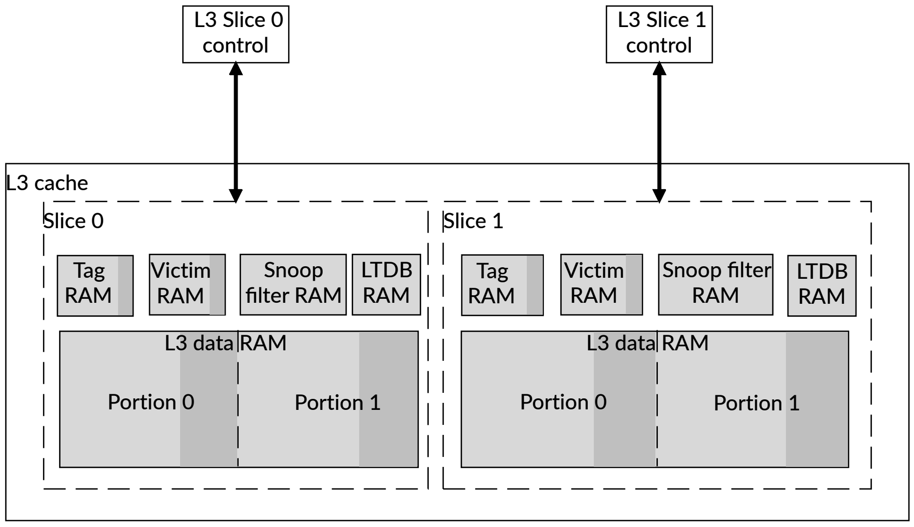
When selecting a non-power-of-two L3 cache size of 1.5MB or 3MB, each cache slice is only implemented with 12 ways. Unlike the power-of-two 256KB, 512KB, 1MB, 2MB, or 4MB L3 cache sizes that implement 16 ways.

For a non-power-of-two L3 cache size:

- The L3 cache size that is reported to software is the next higher power-of-two cache size. For instance, software reports the 1.5MB L3 cache size as being 2MB and the 3MB L3 cache size as being 4MB.
- The last ¼ of the tag, victim, and data RAMs are not implemented in each cache slice. Therefore, partially powering down the 1.5MB or 3MB L3 cache from Full to ¾ has no effect on the power consumption of the DSU because these ways are not implemented.

The following diagram shows the 1.5MB or 3MB L3 cache implementation that is compared with a 2MB or 4MB cache implementation respectively.

Figure 6-3: 1.5MB or 3MB L3 cache implementation



▨ Not implemented cache ways
⬜ Implemented for the 12 required ways

For more information on the cache partial powerdown, see 5.4.1 L3 cache partial powerdown on page 39.

7. ACE master interface

This chapter describes the ACE master memory interface.

7.1 About the ACE master interface

You can configure the DynamIQ™ Shared Unit to use the ACE5 protocol for the master memory interface.

7.1.1 Dual ACE interfaces

The DSU can be implemented with one or two ACE interfaces. Two interfaces give greater bandwidth for memory transactions.

Transactions generated by the cluster are routed to either interface based on the transaction type, memory type, and transaction address:

- All DVM transactions are routed to interface 0.
- All Device transactions are routed to interface 0.

For Cacheable transactions, the configuration parameter `INTERLEAVE_ADDR_BIT` controls which transaction address bit is used to select the routing between interface 0 and interface 1. The default value is to select bit 6 of the transaction address, which interleaves on cache line boundaries. If the selected transaction address bit has a value 0, then interface 0 is used, otherwise interface 1 is used. For more information on `INTERLEAVE_ADDR_BIT` and the impact on performance, see 2.3 Implementation options on page 18.

For Normal Non-cacheable transactions, routing is dependent on the value of CLUSTERECTLR.Non-cacheable behavior control, bit[0]:

- If CLUSTERECTLR[0] is set to 1, Normal Non-cacheable transactions are routed to both interface 0 and 1 in the same way as Cacheable transactions, using the same bit of the transaction address.
- If CLUSTERECTLR[0] is set to 0, Normal Non-cacheable transactions are routed to interface 0.

Note

Setting CLUSTERECTLR.Non-cacheable behavior control has other implications for the system. See 11.8 CLUSTERECTLR, Cluster Extended Control Register on page 105 for more details.

When the external memory system sends snoops, it must either:

- Send the snoop to both interfaces.

- Send the snoop only to the interface that is relevant for the address of that snoop. This behavior is normal operation for an external memory system that contains a snoop filter. The snoop filter indicates that the line is present in one of the two masters.

The second method is more efficient, and if two masters are implemented, Arm recommends that the external memory system includes a snoop filter.

DVM messages

DVM messages can be received on both interfaces, however they are only required on interface 0, and any DVM message sent to interface 1 is treated as a no-op. Therefore for best performance, Arm recommends that, when possible, your interconnect is configured to avoid sending DVM messages to interface 1.

7.2 ACE configurations

The following table shows the supported ACE configurations.

Table 7-1: Supported ACE configurations

Signal	Feature			
	ACE non-coherent		ACE coherent	
	With no cache or invisible system cache	With visible system cache	With no cache or invisible system cache	With visible system cache
BROADCASTCACHEMAINT	0	1	0	1
BROADCASTOUTER	0	0	1	1

Note

- ACE non-coherent mode, no system cache, can be used to connect to an AXI interconnect.
- A visible system cache requires cache maintenance transactions to ensure that a write is visible to all observers.
- An invisible system cache is one that does not require cache maintenance transactions to ensure that a write is visible to all observers. This is true even if those observers use different memory attributes.

The following table shows the key features in each of the supported ACE configurations.

Table 7-2: Supported features in the ACE configurations

Features	DSU Configuration		
	ACE non-coherent No system cache	ACE non-coherent System cache	ACE coherent
AXI3 or AXI4 interconnect compliance	Yes	No	No
ACE interconnect compliance	Yes	Yes	Yes
Barriers on AR and AW channels	No	No	No

Features	DSU Configuration		
	ACE non-coherent No system cache	ACE non-coherent System cache	ACE coherent
Cache maintenance requests on AR channel	No	Yes	Yes
Snoops on AC channel	No	No	Yes
Coherent requests on AR or AW channel	No	No	Yes
DVM requests on AR channel	No	No	Yes

7.3 ACE features

AMBA defines a set of interface properties for the ACE interconnect. The following table shows which of these properties the DSU supports, or requires the cluster interconnect and system to support.

Table 7-3: ACE interconnect properties for the DSU

ACE property	Supported by the DSU	Interconnect support required
Continuous_Cache_Line_Read_Data	Not applicable	Yes
Multi_Copy_Atomicity	Yes	Yes
Ordered_Write_Observation	Not applicable	No
WriteEvict_Transaction	Yes if CLUSTERECTLR Cache UniqueClean eviction control is programmed to 1.	Yes if CLUSTERECTLR Cache UniqueClean eviction control is programmed to 1.
DVM_v8	Yes	Yes if BROADCASTOUTER is HIGH.
Atomic_Transactions	Yes, for atomic instructions that access cacheable memory. No, for atomic instructions that access Device or non-cachable memory.	No
DVM_v8.1	Yes	Yes if BROADCASTOUTER is HIGH.
Cache_Stash_Transactions	No	No
DeAllocation_Transactions	No	No
Persistent_CMO	No	No
Poison	No	No
Data_Check	No	No
QoS_Accept	No	No
Trace_Signals	No	No
Loopback_Signals	No	No
Low_Power_Signals	Yes	Yes
Untranslated_Transactions	No	No
NSAccess_Identifiers	No	No

7.4 ACE master interface attributes

This section describes the read and write issuing capabilities and ID encoding.

The following table lists the read and write issuing capabilities.

Table 7-4: ACE master interface attributes

Attribute	Value	Comments
Write issuing capability	Configuration dependent	The maximum number of writes is: • 96, if two slices are present. • 32, if one slice is present. See 6.7 Cache slices and portions on page 62. Device and Normal Non-cacheable transactions are limited to a total of 15 write transactions by default. This value can be used by system components to size buffers when bridging to other interface protocols, for example PCIe. Normal Non-cacheable transactions can be removed from this limit by setting the CLUSTERECTLR.Noncacheable behavior control, bit[0] to 1. See 11.8 CLUSTERECTLR, Cluster Extended Control Register on page 105.
Read issuing capability	Configuration dependent	The maximum number of reads is: • 98, if two slices are present. • 34, if one slice is present. **Note:** • Two-part *Distributed Virtual Memory* (DVM) messages use the same ID for both parts, and therefore can have two outstanding transactions on the same ID. • For Device and Normal Non-cacheable reads, the read issuing capability is limited by the combined issuing capability. Because the combined issuing capability is always lower than the read issuing capability.
Combined issuing capability	Configuration dependent	The combined issuing capability is: • 98, if two slices are present. • 34, if one slice is present. The Device combined issuing capability is limited to: • 39, if two slices are present. • 23, if one slice is present. The Device and Normal Non-cacheable combined issuing capability is limited to: • 78, if two slices are present and the CLUSTERECTLR.Noncacheable behavior control, bit[0] is set to 1. • 39, if two slices are present and the CLUSTERECTLR.Noncacheable behavior control, bit[0] is set to 0. • 23, if one slice is present.
Exclusive access thread capability	Number of hardware threads	Each hardware thread can have 1 exclusive access sequence in progress.

Attribute	Value	Comments
Write ID capability	Configuration dependent	The maximum write ID capability is: • 96, if two slices are present. • 32, if one slice is present. Only Device memory types with nGnRnE or nGnRE can have more than one outstanding transaction with the same AXI ID. All other memory types use a unique AXI ID for every outstanding transaction.
Write ID width	8	The ID encodes the source of the memory transaction. See the Encodings for AWIDM0[7:0] table.
Read ID capability	Configuration dependent	The maximum read ID capability is: • 98, if two slices are present. • 34, if one slice is present. Only Device memory types with nGnRnE or nGnRE can have more than one outstanding transaction with the same AXI ID. All other memory types use a unique AXI ID for every outstanding transaction. Two part DVMs use the same ID for both parts, and therefore can have two outstanding transactions on the same ID.
Read ID width	9	The ID encodes the source of the memory transaction. See the Encodings for ARIDM0[8:0] table.

Note

The issuing capability described here is the maximum possible for the whole cluster. This can be used to size interconnect capabilities if you want to achieve the maximum performance available. However, this maximum may not be able to be reached by a single core on its own. It may need multiple cores generating heavy memory traffic simultaneously to reach it. The capabilities will vary by core type, for example big cores will typically generate more transactions than LITTLE cores. It can also vary by memory type, with typically a significantly lower limit for Device or Non-cacheable transactions than for Cacheable transactions.

The following table shows the encodings for AWIDM0[7:0], WIDM0[7:0]. When two ACE masters are configured, the maximum number of reads and writes are unchanged. The reads and writes can be distributed between the two masters, or all send to one of the masters, depending on the memory type and address.

Table 7-5: Encodings for AWIDM0[7:0] and WIDM0[7:0]

Attribute	Value	Issuing capability per ID	Comments
Write ID	`0b000t0nnn`	1	Core `nnn`[3], thread `t`[4], system domain store exclusives (except for those that are Device non-reorderable).
	`0b001t0nnn`	15	Core `nnn`[3] thread `t`[4], non-reorderable Device writes.
	`0b1xxxxxxx`[5]	1	All other types of write.
	Other encodings	-	Not used

The following table shows the Encodings for ARIDM0[8:0].

Table 7-6: Encodings for ARIDM0[8:0]

Attribute	Value	Issuing capability per ID	Comments
Read ID	`0b0000t0nnn`	1	Core `nnn`[3], thread `t`[4], load exclusives (except for those that are Device non-reorderable), and Cacheable Shareable store exclusives (sent as exclusive CleanUnique transactions).
	`0b0001t0nnn`	17	Core `nnn`[3], thread `t`[4], non-reorderable Device reads.
	`0b001000000`	1	DVM Sync
	`0b001000001`	256	DVM Complete
	`0b01xxxxxxx`[5]	1	All other types of read
	`0b1xxxxxxxx`[5]		
	Other encodings	-	Not used

Note

- These ID and transaction details are provided for information only. Arm strongly recommends that all interconnects and peripherals are designed to support any type and number of transactions on any ID, to ensure compatibility with future products.
- The Device and Normal Non-cacheable transaction limits that are specified in Table 7-4: ACE master interface attributes on page 69 apply.

For more information about the ACE and AXI signals that are described in this manual, see the *AMBA® AXI and ACE Protocol Specification* .

Related information

Cache slices and portions on page 62

[3] `nnn` is the core number `0b000`-`0b111` in binary.
[4] `t` is the hardware thread number and is 0 if the core does not support multiple hardware threads.
[5] `x` is a do not care value, can be 0 or 1.

7.5 ACE channel properties

The following table shows the properties of the ACE channels.

Table 7-7: ACE channel properties

Property	Value	Comment
Snoop acceptance capability	9 per master interface	The SCU can accept and process a maximum of nine snoop requests from the system for each ACE master interface. It counts requests from the request being accepted on the AC channel to the response being accepted on the CR channel.
Snoop latency	Hit	When there is a hit in L3 cache, the best case for response and data is 10 SCLK cycles. When there is a miss in the L3 cache but a hit in an L1 or L2 cache in a core, then the latency varies depending on the type and configuration of the core. **Note:** Latencies can be higher if hazards occur or if there are not enough buffers to accept requests.
	Miss	Best case six SCLK cycles when the snoop filter and L3 cache tags indicate the miss.
	DVM	The cluster takes a minimum of six SCLK cycles to provide a response to DVM packets.
Snoop filter	Supported	The cluster provides support for an external snoop filter in an interconnect. It indicates when clean lines are evicted from the cluster by sending Evict transactions on the write channel. However there are some cases that can prevent an Evict transaction from being sent. Therefore you must ensure that you build any external snoop filter to handle a capacity overflow. When exceeding capacity, the snoop filter should send a back-invalidation to the cluster. Examples of cases where evicts are not produced include: • Linefills that take external aborts. • Store exclusives that fail. • Mis-matched aliases.
Supported transactions	-	All transactions described by the ACE protocols: • Are accepted on the master interface from the system. • Can be produced on the ACE master interface except: ◦ Barriers ◦ MakeInvalid ◦ ReadShared ◦ ReadOnceCleanInvalid ◦ ReadOnceMakeInvalid ◦ StashOnceShared ◦ StashOnceUnique ◦ StashTranslation ◦ WriteUniquePtl.[6] ◦ WriteUniqueFull.[6] ◦ WriteUniquePtlStash ◦ WriteUniqueFullStash

See the *AMBA® AXI and ACE Protocol Specification* for more information about the ACE channel.

7.6 ACE transactions

The DSU does not generate any FIXED bursts and a burst does not cross a cache line boundary.

The cache linefill fetch length is always 64 bytes.

The DSU generates only a subset of all possible ACE transactions on the master interface.

For WriteBack Cacheable transfers, the supported transfers are:

- WRAP 4 128-bit for read transfers (linefills).
- INCR 4 128-bit for write transfers (evictions).
- INCR 4 128-bit for read transfers (linefills).
- INCR 1 128-bit for read transfers if ACP is configured.

For Normal Non-cacheable or Device transactions:

- INCR N (N:2 or 4) 128-bit read transfers.
- INCR N (N:2 or 4) 128-bit write transfers.
- WRAP N (N:2 or 4) 128-bit read transfers.
- INCR 1 8-bit, 16-bit, 32-bit, 64-bit, and 128-bit read transfers.
- INCR 1 8-bit, 16-bit, 32-bit, 64-bit, and 128-bit write transfers.
- INCR 1 8-bit, 16-bit, 32-bit, 64-bit, and 128-bit exclusive read transfers.
- INCR 1 8-bit, 16-bit, 32-bit, 64-bit, and 128-bit exclusive write transfers.

The following points apply to ACE transactions:

- WRAP bursts are only 128-bit size.
- INCR burst, more than one transfer, are only 128-bit size.
- No transaction is marked as FIXED.
- Write transfers with none, some, or all byte strobes LOW can occur.

The following table shows the ACE transactions that can be generated and some typical operations that might cause these transactions to be generated.

[6] The AMBA ACE5 transaction types WriteUniqueFull and WriteUniquePtl were known in AMBA4 ACE as WriteLineUnique and WriteUnique, respectively.

Note

This table does not provide an exhaustive list of operations that generate each type of transaction, because there are many possibilities.

Table 7-8: ACE transactions

Transaction	Operation
ReadNoSnoop	Non-cacheable loads or instruction fetches. Linefills of non-shareable cache lines into L1, L2, or L3 caches.
ReadOnce	Cacheable loads that are not allocating into the cache.
ReadClean	Cache data linefills started by a load instruction. Cache linefills started by an instruction fetch.
ReadNotSharedDirty	Cache data linefills started by a load instruction. Cache linefills started by an instruction fetch.
ReadUnique	Data linefills started by a store instruction.
CleanUnique	Store instructions that hit in the cache but the line is not in a unique coherence state. Store instructions that are not allocating into the caches, for example when streaming writes.
MakeUnique	Store instructions of a full cache line of data, that miss in the caches.
CleanShared	Cache maintenance instructions.
CleanSharedPersist	Not used.
CleanInvalid	Cache maintenance instructions.
DVM	TLB and instruction cache maintenance instructions.
DVM Complete	DVM Sync snoops received from the interconnect.
WriteNoSnoop	Non-cacheable store instructions. Evictions of non-shareable cache lines from L1, L2, and L3 caches.
WriteBack	Evictions of dirty lines from the L1, L2, or L3 cache. Streaming writes that are not allocating into the cache.
WriteClean	Evictions of dirty lines from the L3 cache, when the line is still present in an L1 or L2 cache. Some cache maintenance instructions.
WriteEvict	Evictions of unique clean lines, when configured in the CLUSTERECTLR.
Evict	Evictions of clean lines, when configured in the CLUSTERECTLR.
Barriers	Not used.
MakeInvalid	Not used.
ReadShared	Not used.
ReadOnceCleanInvalid	Not used.
ReadOnceMakeInvalid	Not used.
StashOnceShared	Not used.

Transaction	Operation
StashOnceUnique	Not used.
StashTranslation	Not used.
WriteUniquePtl	Not used.
WriteUniqueFull[7]	Not used.
WriteUniquePtlStash	Not used.
WriteUniqueFullStash	Not used.

7.7 Support for memory types

The cores in the DynamIQ™ cluster simplify the coherency logic by downgrading some memory types.

Normal memory that is marked as both Inner Write-Back Cacheable and Outer Write-Back Cacheable is cached in the core data caches and the L3 cache.

All other Normal memory types are treated as Non-cacheable and are sent on the master interface as Normal Non-cacheable.

7.8 Read response

The ACE master can delay accepting a read data channel transfer by holding RREADY LOW for an indeterminate number of cycles.

RREADY can be deasserted LOW between read data channel transfers that form part of the same transaction.

The ACE master asserts the read acknowledge signal RACK HIGH in the ACLK cycle following acceptance of the last read data channel transfer for a transaction. RACK is asserted in AXI compatibility mode in addition to ACE configurations.

Note

- For interoperability of system components, Arm recommends that components interfacing with the ACE master are fully ACE-compliant with no reliance on the subset of permitted RACK behavior that is described for the DSU.
- If the interconnect does not perform hazarding between coherent and non-coherent requests, then, after it has returned the first transfer of read data for a non-coherent read, it must return all the remaining read transfers in the transaction.

[7] The AMBA5 ACE transaction types WriteUniqueFull and WriteUniquePtl were known in AMBA4 ACE as WriteLineUnique and WriteUnique, respectively.

The completion of the read transfers must not depend on either of the following:

- Snoop requests being sent to the core.
- The core needing to respond to a snoop request that could be to the same address.

7.9 Write response

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

The ACE master always accepts write responses without delay by holding BREADY HIGH.

The ACE master asserts the write acknowledge signal WACK HIGH in the ACLK cycle following acceptance of a write response. WACK is asserted in AXI compatibility mode in addition to ACE configurations.

Note

For interoperability reasons, Arm recommends that system components fully comply with the ACE specification and do not rely on the DSU behavior described here.

7.10 Barriers

The DSU does not support sending barrier transactions to the interconnect. Barriers are always terminated within the cluster.

You must ensure that your interconnect and any peripherals connected that are to it do not return a write response for a transaction until that transaction would be considered complete by a later barrier. This means that the write must be observable to all other masters in the system. Arm expects most peripherals to meet this requirement.

7.11 AXI compatibility mode

The DSU implements an AXI compatibility mode that enables you to use the DSU in a standalone environment where the AMBA ACE5 interface is not required.

To enable this mode, you must ensure that the BROADCASTOUTER, BROADCASTCACHEMAINT, and BROADCASTPERSIST input signals are set to LOW. You must also tie the ACVALIDMx, ACWAKEUPMx, CDREADYMx, and CRREADYMx input signals LOW.

The AXI3 protocol supports write interleaving which is not used by the DSU. To allow compatibility with AXI3 components, the DSU provides WIDMx output signals, which can be connected to an AXI3 device.

If using AXI4 and ACE components, you must leave the WIDMx output signals unconnected as these signals do not exist in these protocols.

For single master implementations, WIDM0 is provided. For dual master implementations, WIDM0 and WIDM1 are provided.

7.11.1 Additional logic to support AXI compatibility

To support AXI compatibility, the DSU requires additional logic.

The DSU implements a handshake for system coherency using the SYSCOREQ* and SYSCOACK* signals. In AXI compatibility mode, the DSU does not support coherency. However, because this interface uses a handshake protocol, transitions on SYSCOREQ* request must be responded to by a corresponding acknowledge on SYSCOACK*.

Arm recommends that you implement the following logic in your system:

- Add a single-bit register with the input connected to SYSCOREQ*.
- Connect the output of the register to SYSCOACK*.
- The register must be reset LOW, and clocked and reset using the clock and reset from your AXI system.

For single ACE master implementations, SYSCOREQM0 and SYSCOACKM0 are provided. For dual ACE master implementations, SYSCOREQM1 and SYSCOACKM1 are also provided.

7.12 ACE privilege information

ACE provides information about the privilege level of accesses on the ARPROTM[0] and AWPROTM[0] signals. This information is not available from cores within the cluster. Therefore these signals are always driven to HIGH indicating that the access could be a privileged access.

8. CHI master interface

This chapter describes the AMBA 5 CHI master memory interface.

8.1 About the CHI master interface

You can configure the DSU to use the AMBA 5 CHI protocol for the master memory interface.

8.1.1 Dual CHI interfaces

The DSU can be implemented with one or two CHI interfaces.

Transactions generated by the cluster are routed to either interface based on the transaction type, memory type, and transaction address:

- All DVM transactions are routed to interface 0.
- All Device transactions are routed to interface 0.

For Cacheable transactions, the configuration parameter `INTERLEAVE_ADDR_BIT` controls which transaction address bit is used to select the routing between interface 0 and interface 1. The default value is to select bit 6 of the transaction address, which interleaves on cache line boundaries. If the selected transaction address bit has a value 0, then interface 0 is used, otherwise interface 1 is used. For more information on `INTERLEAVE_ADDR_BIT` and the impact on performance, see 2.3 Implementation options on page 18.

For Normal Non-cacheable transactions, routing is dependent on the value of CLUSTERECTLR.Non-cacheable behavior control, bit[0]:

- If CLUSTERECTLR[0] is set to 1, Normal Non-cacheable transactions are routed to both interface 0 and 1 in the same way as Cacheable transactions, using the same bit of the transaction address.
- If CLUSTERECTLR[0] is set to 0, Normal Non-cacheable transactions are routed to interface 0.

When the external memory system sends snoops, it must either:

- Send the snoop to both interfaces.
- Send the snoop only to the interface that is relevant for the address of that snoop. This behavior is normal operation for an external memory system that contains a snoop filter. The snoop filter indicates that the line is present in one of the two masters.

The second method is more efficient, and if two masters are implemented, Arm recommends that the external memory system includes a snoop filter.

DVM messages

DVM messages can be received on both interfaces, however they are only required on interface 0, and any DVM message sent to interface 1 is treated as a no-op. Therefore for best performance, Arm recommends that, when possible, your interconnect is configured to avoid sending DVM messages to interface 1.

System address map

If the DSU is configured with the integrated CMN-600 SAM and two CHI interfaces, the DSU will only contain one SAM, and the node_id field in the CMN-600 por_rnsam_node_info register will contain the node ID of CHI interface 0.

8.2 CHI version

The DSU supports CHI Issue B, Issue C, or Issue D, depending on configuration.

Direct Connect configurations support CHI Issue C or Issue D, depending on core type, and all other configurations only support CHI Issue B.

The widths of the TXDAT and RXDAT channels are increased by one bit when CHI Issue C is enabled. CHI Issue D affects the widths of all channels.

8.3 CHI features

AMBA defines a set of interface properties for the CHI interconnect. The following table shows which of these properties the DSU supports, or requires the interconnect and system to support.

Table 8-1: CHI interconnect properties for the DSU

CHI property	Supported by the DSU	Interconnect support required
Atomic_Transactions	Yes if BROADCASTATOMIC is HIGH.	Yes if BROADCASTATOMIC is HIGH.
Cache_Stash_Transactions	Yes	Yes
Direct_Memory_Transfer	Yes	Optional. The DSU supports this feature if it is implemented by the interconnect.
Direct_Cache_Transfer	Yes	Optional. The DSU supports this feature if it is implemented by the interconnect.
Data_Poison	Yes if cache protection (`SCU_CACHE_PROTECTION`) is enabled.	Yes if cache protection (`SCU_CACHE_PROTECTION`) is enabled.
Data_Check	No	No
CCF_Wrap_Order	Yes. The DSU always sends Data packets in critical chunk first wrap order.	No
Req_Addr_Width	44. If a core with a 48-bit physical address width is configured inside the cluster, this is 48.	Not applicable
NodeID_Width	11	Not applicable
Data_Width	User configurable: 128 bits or 256 bits.	Not applicable
Barrier_Transactions	No	No. The DSU does not use these transaction types.

CHI property	Supported by the DSU	Interconnect support required
Data return from SC state.	Yes	Not applicable
I/O de-allocation transactions (ROMI and ROCI).	No	No. The DSU does not use these transaction types.
ReadNotSharedDirty transactions	Yes	Yes
CleanSharedPersist transactions	Yes if BROADCASTPERSIST is HIGH.	Yes if BROADCASTPERSIST is HIGH.

For more information on these features, see *AMBA® 5 CHI Architecture Specification*.

8.4 CHI configurations

You can change the coherency configurations to suit your system configuration using the BROADCASTCACHEMAINT and BROADCASTOUTER input signals.

The following table shows the permitted combinations of these signals and the supported configurations in the DSU, with a CHI bus.

Table 8-2: Supported CHI configurations

Signal	Feature			
	CHI non-coherent		CHI coherent	
	With no cache or invisible system cache	With visible system cache	With invisible system cache	With visible system cache
BROADCASTCACHEMAINT	0	1	0	1
BROADCASTOUTER	0	0	1	1

Note

- A visible system cache requires cache maintenance transactions to ensure that a write is visible to all observers.
- An invisible system cache is one that does not require cache maintenance transactions to ensure that a write is visible to all observers. This is true even if those observers use different memory attributes.

The following table shows the key features in each of the supported CHI configurations.

Table 8-3: Supported features in the CHI configurations

Features	Configuration		
	CHI non-coherent		CHI coherent
	With no cache or invisible system cache	With visible system cache	
Cache maintenance requests on TXREQ channel	No	Yes	Yes
Snoops on RXSNP channel	No	No	Yes
Coherent requests on TXREQ channel	No	No	Yes
DVM requests on TXREQ channel	No	No	Yes

8.5 Attributes of the CHI master interface

The following table lists the possible values for the read and write issuing capabilities.

Table 8-4: Attributes of the CHI master memory interface

Attribute	Value	Comments
Write issuing capability	Configuration dependent	The maximum number of writes is: • 96, if two slices are present. • 32, if one slice is present.
Read issuing capability	Configuration dependent	The maximum number of reads is: • 96, if two slices are present. • 32, if one slice is present. **Note:** For Device and Normal Non-cacheable reads, the read issuing capability is limited by the combined issuing capability. Because the combined issuing capability is always lower than the read issuing capability.
Combined issuing capability	Configuration dependent	The combined issuing capability is: • 96, if two slices are present. • 32, if one slice is present. The Device combined issuing capability is limited to: • 39, if two slices are present. • 23, if one slice is present. The Device and Normal Non-cacheable combined issuing capability is limited to: • 78, if two slices are present with one main master port. • 78, if two slices are present with two main master ports and the CLUSTERECTLR.Noncacheable behavior control, bit[0] is set to 1. See 11.8 CLUSTERECTLR, Cluster Extended Control Register on page 105. • 39, if two slices are present with two main master ports and the CLUSTERECTLR.Noncacheable behavior control, bit[0] is set to 0. • 23, if one slice is present.
Exclusive hardware access thread capability	Number of hardware threads.	Each hardware thread can have one exclusive access sequence in progress.
Transaction ID width	8 bits	There is no fixed mapping between CHI transaction IDs and cores. Transaction IDs can be used for either reads or writes. **Note:** The source of the transaction is encoded in the LPID field, see Table 8-6: CHI LPID assignment on page 84.

Attribute	Value	Comments
Transaction ID capability	Configuration dependent	The ID capability is: • 96, if two slices are present. • 32, if one slice is present. **Note:** Unlike in an AMBA ACE5 configuration, there is never any ID reuse in CHI implementations, regardless of the memory type.
NodeID widths	11 bits	-
TXREQFLIT.RSVDC	0 bits	-
TXDATFLIT.RSVDC	0 bits	-
TXDATFLIT.DataCheck	0 bits	-

Note

The issuing capability described here is the maximum possible for the whole cluster, and can be used to size interconnect capabilities if you want to achieve the maximum performance available. However this maximum may not be able to be reached by a single core on its own, it may need multiple cores generating heavy memory traffic simultaneously to reach it. The capabilities will vary by core type, for example big cores will typically generate more transactions than LITTLE cores. It can also vary by memory type, with typically a significantly lower limit for Device or Non-cacheable transactions than for Cacheable transactions.

Related information

Cache slices and portions on page 62

8.6 CHI channel properties

The following table shows the snoop capabilities and other CHI channel properties for the DSU.

Table 8-5: CHI channel properties

Property	Value	Comment
Snoop acceptance capability	Configuration dependent	For dual 256-bit CHI masters, the SCU can accept and process a maximum of 11 snoop requests from each master port. For a single 256-bit CHI master, the SCU can accept and process a maximum of 14 snoop requests from the system. For a 128-bit CHI master, the SCU can accept and process a maximum of 11 snoop requests.
DVM acceptance capability	4	The SCU can accept and process a maximum of four DVM transactions from the system. Each of these four transactions can be a two part DVM message. The interconnect must be configured to never send more than four DVM messages to the cluster, otherwise the system might deadlock.

Property	Value	Comment
Snoop latency	Hit	When there is a hit in L3 cache, the best case for response and data is 10 SCLK cycles. When there is a miss in the L3 cache but a hit in an L1 or L2 cache in a core, then the latency varies. This latency variation depends on the type and configuration of that core. Latencies can be higher if hazards occur or if there are not enough buffers to absorb requests.
	Miss	Best case for latency is six SCLK cycles when the snoop filter and L3 cache tags indicate the miss.
	DVM	The cluster takes a minimum of six SCLK cycles to provide a response to DVM packets.
Snoop filter	Supported	The cluster provides support for an external snoop filter in an interconnect. It indicates when clean lines are evicted from the cluster by sending Evict transactions on the CHI write channel. However there are some cases that can prevent an Evict transaction from being sent. Therefore you must ensure that you build any external snoop filter to handle a capacity overflow. When exceeding capacity, the snoop filter should send a back-invalidation to the cluster. Examples of case where evicts are not produced include: • Linefills that take external aborts. • Store exclusives that fail. • Mis-matched aliases.
Supported transactions	-	All transactions that are described by the CHI protocol: • Are accepted on the CHI master interface from the system. • Can be produced on the CHI master interface except: ◦ ReadShared. ◦ MakeInvalid. ◦ EOBarrier. ◦ ECBarrier. ◦ WriteCleanPtl. ◦ WriteUniquePtl. ◦ WriteBackPtl. ◦ WriteUniqueFullStash. ◦ WriteUniquePtlStash. ◦ ReadOnceCleanInvalid. ◦ ReadOnceMakeInvalid.

8.7 CHI transactions

CHI transactions are sent to a specific node in the interconnect that is based on the following criteria:

- Type of access.
- Address of the access.
- Settings of the System Address Map.

Addresses that map to an HN-F node can be marked as cacheable memory in the translation tables, and can take part in the cache coherency protocol. Addresses that map to an HN-I or MN must be marked as device or Non-cacheable memory.

CHI TXREQ transactions include the *Logical Processor ID* (LPID) field. This field uniquely identifies the logical core that generated the request transaction. The following table shows CHI LPID assignment.

Table 8-6: CHI LPID assignment

LPID	Description
`0x0-0xF`	Bits [3:0] are encoded: **[0]** Thread number. **[3:1]** CPUID.
`0x10-0x1D`	Reserved.
`0x1E`	ACP request.
`0x1F`	Cache copy back.

The following table shows the CHI transactions that can be generated and some typical operations that might cause these transactions to be generated.

Note

This table does not provide an exhaustive list of operations that generate each type of transaction, because there are many possibilities.

Table 8-7: CHI transaction types

Transaction	Operation
ReadNoSnp	Non-cacheable loads or instruction fetches. Linefills of Non-shareable cache lines into L1, L2, or L3 caches.
ReadOnce	Cacheable loads that are not allocating into the cache.
ReadClean	Cache data linefills started by a load instruction. Cache linefills started by an instruction fetch.
ReadShared	Not used.
ReadNotSharedDirty	Cache data linefills started by a load instruction. Cache linefills started by an instruction fetch.
ReadUnique	Cache data linefills started by a store instruction.
CleanUnique	Store instructions that hit in the cache but the line is not in a unique coherence state.
MakeUnique	Store instructions of a full cache line of data, that miss in the caches.
CleanShared	Cache maintenance instructions.
CleanSharedPersist	Cache maintenance instructions. The *Data Cache Clean to the Point of Persistence* (`DC CVAP`) cache maintenance instruction only generates this transaction when the BROADCASTPERSIST input signal is HIGH.
CleanInvalid	Cache maintenance instructions.

Transaction	Operation
MakeInvalid	Not used.
DVMOp	TLB and instruction cache maintenance instructions.
EOBarrier	Not used.
ECBarrier	Not used.
PrefetchTgt	Hardware prefetch hint to the memory controller.
StashOnceShared	Cache prefetch when there is no L3 cache present.
StashOnceUnique	Cache prefetch when there is no L3 cache present.
WriteNoSnpPtl	Non-cacheable store instructions.
WriteNoSnpFull	Non-cacheable store instructions. Evictions of Non-shareable cache lines.
WriteUniqueFull	Cacheable writes of a full cache line, that are not allocating into L1, L2, or L3 caches, for example streaming writes.
WriteUniquePtl	Not used.
WriteBackFull	Evictions of dirty lines from the L1, L2, or L3 caches.
WriteBackPtl	Not used.
WriteCleanFull	Evictions of dirty lines from the L3 cache, when the line is still present in an L1 or L2 cache. Some cache maintenance instructions.
WriteCleanPtl	Not used.
WriteEvictFull	Evictions of unique clean lines, when configured in the CLUSTERECTLR.
Evict	Evictions of clean lines, when configured in the CLUSTERECTLR.
AtomicStore	Atomic instruction.
AtomicLoad	Atomic instruction.
AtomicSwap	Atomic instruction.
AtomicCompare	Atomic instruction.
WriteUniqueFullStash	Not used.
WriteUniquePtlStash	Not used.
ReadOnceCleanInvalid	Not used.
ReadOnceMakeInvalid	Not used.

External memory accesses generate the following transactions in an implementation configured with a CHI master interface.

Table 8-8: CHI transaction usage

Attributes		CHI transaction				
Memory type	Shareability	SnpAttr	Load	Store	Load exclusive	Store exclusive
Device	Outer Shareable	Non-snoopable	ReadNoSnp	WriteNoSnp	ReadNoSnp and Excl set to HIGH.	WriteNoSnp and Excl set to HIGH.

Attributes		CHI transaction				
Memory type	**Shareability**	**SnpAttr**	**Load**	**Store**	**Load exclusive**	**Store exclusive**
Normal, Inner Non-cacheable, Outer Non-cacheable	Non-shareable	Non-snoopable	ReadNoSnp	WriteNoSnp	ReadNoSnp and Excl set to HIGH.	WriteNoSnp and Excl set to HIGH.
	Inner Shareable					
	Outer Shareable					
Normal, Inner Non-cacheable, Outer Write-Back or Write-Through, or Normal, Inner Write-Through, Outer Write-Back, Write-Through or Non-cacheable, or Normal Inner Write-Back Outer Non-cacheable or Write-Through	Non-shareable	Non-snoopable	ReadNoSnp	WriteNoSnp	ReadNoSnp and Excl set to HIGH.	WriteNoSnp and Excl set to HIGH.
	Inner Shareable					
	Outer Shareable					
Normal, Inner Write-Back, Outer Write-Back	Non-shareable	Non-snoopable	ReadNoSnp	WriteNoSnp when the line is evicted or if not allocating into the cache.	ReadNoSnp	WriteNoSnp when the line is evicted.
	Inner Shareable	Snoopable	ReadNotSharedDirty or ReadClean	ReadUnique, CleanUnique, or MakeUnique if allocating into the cache, then a WriteBackFull when the line is evicted. WriteUniqueFull if not allocating into the cache.	ReadNotSharedDirty or ReadClean, with Excl set to HIGH.	CleanUnique with Excl set to HIGH if required, then a WriteBackFull when the line is evicted.
	Outer Shareable	Snoopable				

8.8 Use of DataSource

Some CHI responses from the interconnect include a DataSource field indicating where the data was supplied from. When making use of the DataSource field, Arm recommends providing this information as accurately as possible using the encodings recommended in the table *Suggested DataSource value encodings* provided in the *AMBA® 5 CHI Architecture Specification*.

The value of this field is used to calculate some PMU events, and can also be used by some cores to tune the performance of their data prefetchers.

8.9 Support for memory types

The cores in the DynamIQ™ cluster simplify the coherency logic by downgrading some memory types.

Normal memory that is marked as both Inner Write-Back Cacheable and Outer Write-Back Cacheable is cached in the core data caches and the L3 cache.

All other Normal memory types are treated as Non-cacheable and are sent on the master interface as Normal Non-cacheable.

9. ACP slave interface

This chapter describes the ACP slave interface.

9.1 About the ACP

The *Accelerator Coherency Port* (ACP) is an optional slave interface, conforming to a subset of the ACE5-Lite specification.

The ACP slave interface allows an external master to access memory through the main memory interface of the DSU. Only access to Cacheable memory is permitted.

The read and write data buses of the ACP are 128 bits. Accesses are optimized for cache line length.

To maintain cache coherency, accesses are checked in all cached locations in the cluster. That is, the L3 cache, and the data caches in each core.

By default, ACP write accesses are implicit stash requests to the L3 cache. Alternatively, implicit stash requests can target the L2 cache of a selected core.

9.2 ACP features

The ACP interface for the DSU supports the following properties.

Table 9-1: ACP interface properties for the DSU

ACP property	Supported by the DSU
Port_Type	Accelerator
Continuous_Cache_Line_Read_Data	Yes
Multi_Copy_Atomicity	Yes
Ordered_Write_Observation	No
WriteEvict_Transaction	No
DVM_v8	No
Atomic_Transactions	No
DVM_v8.1	No
Cache_Stash_Transactions	Yes
DeAllocation_Transactions	No
Persistent_CMO	No
Poison	No
Data_Check	No
QoS_Accept	No

ACP property	Supported by the DSU
Trace_Signals	No
Loopback_Signals	No
Low_Power_Signals	Yes
Untranslated_Transactions	No
NSAccess_Identifiers	No

9.3 ACP ACE5-Lite subset

The ACP conforms to a subset of the ACE5-Lite specification.

The ACP ACE5-Lite subset is described in *AMBA® AXI and ACE Protocol Specification* . The DSU has the following additional restrictions:

- The values of ARCACHES and AWCACHES are restricted to:
    - `0b0111`.
    - `0b1011`.
    - `0b1111`.
- All transactions can be Secure or Non-secure.
- Exclusive accesses are not supported. ARLOCK and AWLOCK signals are not present.
- All requests can specify Outer Shareable and Non-shareable using the AWDOMAINS and ARDOMAINS signals.
- Barriers are not supported. The BRESP response for any write transaction indicates global observability for the transaction.
- ARSIZE and AWSIZE signals are not present. A value of `4` (16 bytes) is assumed.
- The values of ARLENS and AWLENS are restricted to:

0	One beat.
3	Four beats.

- ARBURST and AWBURST signals are not present. A value of `0b01` (INCR) is assumed.
- ARSNOOP signals are not present. A value of `0b0000` is assumed.
- ARQOS and AWQOS signals are not present.

9.4 ACP transaction types

The ACP supports transaction types having the following transfer size and length combinations:

- 16-byte INCR read transaction:
    - ARLENS is 0 (one beat).

 - Address aligned to 16-byte boundary (ARADDRS[3:0] is `0b0000`).
- 64-byte INCR read transaction:
    - ARLENS is 3 (four beats).
    - Address aligned to 64-byte boundary (ARADDRS[5:0] is `0b000000`).

The ACP supports the following write transaction transfer size and length combinations:

- 16-byte INCR write transaction:
    - WSTRBS, any combination of bytes, including no bytes, are valid.
    - AWLENS is 0 (one beat).
    - AWSNOOPS is WriteUniquePtl or WriteUniquePtlStash.
    - Address aligned to 16-byte boundary (AWADDRS[3:0] is `0b0000`).
- 64-byte INCR write transaction:
    - WSTRBS, any combination of bytes, including no bytes, are valid.

      Note

      When AWSNOOPS is WriteUniqueFull, all bytes must be valid.

    - AWLENS is 3 (four beats).
    - AWSNOOPS is WriteUniquePtl, WriteUniquePtlStash, or WriteUniqueFull.
    - Address aligned to 64-byte boundary (AWADDRS[5:0] is `0b000000`).

Note

- The AMBA 5 ACE-Lite transaction types WriteUniqueFull and WriteUniquePtl were known in AMBA 4 ACE-Lite as WriteLineUnique and WriteUnique, respectively.
- The DSU treats WriteUniquePtlStash as a WriteUniquePtl and does not perform a stash operation for this transaction type.

The ACP supports the following Cache Stash Transaction transfer size and length:

- 64-byte INCR write stash transaction:
    - WSTRBS, all bytes are valid.
    - AWLENS is 3 (four beats).
    - AWSNOOPS is WriteUniqueFullStash.
    - Address aligned to 64-byte boundary (AWADDRS[5:0] is `0b000000`).
- Dataless 64-byte INCR write stash transaction:
    - No W-Channel transfers.
    - AWLENS is 3.
    - AWSNOOPS is StashOnceShared or StashOnceUnique.

    ◦ Address aligned to 64-byte boundary (AWADDRS[5:0] is `0b000000`).

Stash requests can target the L2 cache of a selected core by asserting signal AWSTASHLPIDENS and indicating the selected core number on AWSTASHLPIDS[3:1]. The signal AWSTASHLPIDS[0] is reserved for the thread number, but this does not affect the stash request.

Note

Requests not meeting these restrictions cause a SLVERR response on RRESPS or BRESPS.

The following table lists the ACP supported transactions:

Table 9-2: ACP supported transactions

Transaction	Notes
ReadOnce	-
WriteUniqueFull	-
WriteUniquePtl	-
WriteUniquePtlStash	Treated as a WriteUniquePtl. The DSU does not perform a stash.
WriteUniqueFullStash	-
StashOnceUnique	-
StashOnceShared	-

9.5 ACP performance

For optimum performance, use the following guidelines for ACP transactions.

The master must avoid sending more than one outstanding transaction on the same AXI ID, to prevent the second transaction stalling the interface until the first has completed. If the master requires explicit ordering between two transactions, Arm recommends that it waits for the response to the first transaction before sending the second transaction.

Writes are higher performance when they use WriteUniqueFull or WriteUniqueFullStash transactions.

WriteUniquePtl or WriteUniquePtlStash transactions always incur a read-modify write sequence.

Some L3 resources are shared between the ACP interface and the cores. Therefore, heavy traffic on the ACP interface might, in some cases, reduce the performance of the cores.

Write transactions use the Write-Allocate bit of the memory type (AWCACHES[3]) to decide whether to allocate to the L3 cache, as follows:

- If the stash request does not target a core (AWSTASHLPIDENS is LOW) and AWCACHES[3] is HIGH, then the cache line is allocated to the L3 cache.

- If the stash request does not target a core (AWSTASHLPIDENS is LOW) and AWCACHES[3] is LOW, then the cache line is not allocated to the L3 cache and it will be written out on the master port instead.
- When the stash request does not target a core (AWSTASHLPIDENS is LOW), then the WriteUniqueFullStash transaction performs the same operation as WriteUniqueFull.
- Stash requests that target a core (AWSTASHLPIDENS is HIGH) always attempt to allocate to the core L2 cache. In this case, Arm recommends that AWCACHES[3] is HIGH. Since, if AWCACHES[3] is LOW, then the line will not initially be allocated to the cache. Instead the line will be written out on the master port before being fetched back into the core, which is inefficient.

The following table describes the ACP acceptance capabilities.

Table 9-3: ACP acceptance capabilities

Attribute	Value	Description
Write acceptance capability	33	The ACP can accept up to 33 write transactions.
Read acceptance capability	33	The ACP can accept up to 33 read transactions.
Combined acceptance capability	34	The ACP can accept up to 34 transactions. There is no performance benefit above 32 outstanding transactions.

10. AXI master peripheral port

This chapter describes the AXI master peripheral port.

10.1 About the peripheral port

The peripheral port supports Device accesses to tightly coupled accelerators.

The peripheral port can be used for low-latency access to peripherals local to the cluster. It has the same latency as the main master port. However, the overall system latency to devices that are connected to the main master port is greater because of the higher latency of the system interconnect.

The peripheral port is optionally implemented. It is a 64-bit AXI4 master interface.

The peripheral port supports access to only Device-nGRE, nGnRE, and nGnRnE memory types:

- All accesses must be aligned load or store instructions of 64 bits or less. Unaligned or larger accesses are not supported and generate an external abort.
- Atomic instructions are not supported and generate an external abort.
- Load and store exclusive instructions are not supported. Store exclusive instructions will fail and the result register will reflect this, but the memory location might be updated.

Accesses to the port using other memory types are unpredictable.

The peripheral port address range is defined by two configuration input buses. ASTARTMP[PA-1:20] for the start of the address range, which is inclusive, and AENDMP[PA-1:20] for the end of the address range, which is exclusive. PA is the largest physical address width of any connected core. These signals are only captured at reset.

10.2 Transaction ID encoding

The AXI interface provides several read and write issuing capabilities and requires a specific format for AXI transaction IDs.

The following table describes the read and write issuing capabilities of the AXI interface.

Table 10-1: AXI issuing capabilities

Attribute	Value	Comments
Write issuing capability	5	There can be up to 5 outstanding write transactions.
Read issuing capability		There can be up to 5 outstanding read transactions.
Combined issuing capability	10	There can be up to 10 outstanding transactions.
Write ID capability	5	Each ID can have up to 5 outstanding write transactions.
Read ID capability		Each ID can have up to 5 outstanding read transactions.

The following table lists the encoding for AXI transaction IDs.

Table 10-2: AXI transaction ID encoding

Attribute	Value	Comments
All IDs	tnnn	Thread t[8], core nnn[9]

Note

These ID and transaction details are provided for information only. Arm strongly recommends that all interconnects and peripherals are designed to support any type and number of transactions on any ID, to ensure compatibility with future products.

See the *AMBA® AXI and ACE Protocol Specification* for more information about the ACE and AXI signals described in this manual.

[8] t is the hardware thread number and is 0 when the core does not support multiple hardware threads.
[9] nnn is the core number `0b000-0b111` in binary.

11. Control registers

This chapter describes the control registers for the DSU.

11.1 About the control registers

The DSU contains system control registers in the SCU and L3 logic to control the functionality of the cluster. Most of these registers are shared between all the cores in the cluster, but a few are private to each core.

The chapter is presented as follows:

AArch32 control register summary

This section lists the AArch32 control registers by access encoding.

AArch64 control register summary

This section lists the AArch64 control registers by access encoding.

Register descriptions

The remainder of the chapter provides generic register descriptions, that apply to both AArch32 and AArch64 registers. They are listed in alphabetical order.

11.2 AArch32 control register summary

This section lists the AArch32 control registers implemented in the DSU, sorted by access encoding.

Table 11-1: DynamIQ™ Shared Unit AArch32 control registers

Register mnemonic	Copro	CRn	Opc1	CRm	Opc2	Width	Register name and description
CLUSTERCFR	cp15	c15	0	c3	0	32	11.7 CLUSTERCFR, Cluster Configuration Register on page 101
CLUSTERIDR	cp15	c15	0	c3	1	32	11.9 CLUSTERIDR, Cluster Main Revision ID Register on page 109
CLUSTERREVIDR	cp15	c15	0	c3	2	32	11.16 CLUSTERREVIDR, Cluster Revision ID Register on page 125
CLUSTERACTLR	cp15	c15	0	c3	3	32	11.5 CLUSTERACTLR, Cluster Auxiliary Control Register on page 99
CLUSTERECTLR	cp15	c15	0	c3	4	32	11.8 CLUSTERECTLR, Cluster Extended Control Register on page 105
CLUSTERPWRCTLR	cp15	c15	0	c3	5	32	11.13 CLUSTERPWRCTLR, Cluster Power Control Register on page 119
CLUSTERPWRDN	cp15	c15	0	c3	6	32	11.14 CLUSTERPWRDN, Cluster Powerdown Register on page 121
CLUSTERPWRSTAT	cp15	c15	0	c3	7	32	11.15 CLUSTERPWRSTAT, Cluster Power Status Register on page 123

Register mnemonic	Copro	CRn	Opc1	CRm	Opc2	Width	Register name and description
CLUSTERTHREADSID	cp15	c15	0	c4	0	32	11.18 CLUSTERTHREADSID, Cluster Thread Scheme ID Register on page 129
CLUSTERACPSID	cp15	c15	0	c4	1	32	11.4 CLUSTERACPSID, Cluster ACP Scheme ID Register on page 97
CLUSTERSTASHSID	cp15	c15	0	c4	2	32	11.17 CLUSTERSTASHSID, Cluster Stash Scheme ID Register on page 127
CLUSTERPARTCR	cp15	c15	0	c4	3	32	11.12 CLUSTERPARTCR, Cluster Partition Control Register on page 114
CLUSTERBUSQOS	cp15	c15	0	c4	4	32	11.6 CLUSTERBUSQOS, Cluster Bus QoS Control Register on page 99
CLUSTERL3HIT	cp15	c15	0	c4	5	32	11.10 CLUSTERL3HIT, Cluster L3 Hit Counter Register on page 111
CLUSTERL3MISS	cp15	c15	0	c4	6	32	11.11 CLUSTERL3MISS, Cluster L3 Miss Counter Register on page 113
CLUSTERTHREADSIDOVR	cp15	c15	0	c4	7	32	11.19 CLUSTERTHREADSIDOVR, Cluster Thread Scheme ID Override Register on page 130

11.3 AArch64 control register summary

This section lists the AArch64 control registers implemented in the DSU, sorted by access encoding.

Table 11-2: DynamIQ™ Shared Unit AArch64 control registers

Register mnemonic	Op0	CRn	Op1	CRm	Op2	Width	Register name and description
CLUSTERCFR_EL1	3	c15	0	c3	0	32	11.7 CLUSTERCFR, Cluster Configuration Register on page 101
CLUSTERIDR_EL1	3	c15	0	c3	1	32	11.9 CLUSTERIDR, Cluster Main Revision ID Register on page 109
CLUSTERREVIDR_EL1	3	c15	0	c3	2	32	11.16 CLUSTERREVIDR, Cluster Revision ID Register on page 125
CLUSTERACTLR_EL1	3	c15	0	c3	3	32	11.5 CLUSTERACTLR, Cluster Auxiliary Control Register on page 99
CLUSTERECTLR_EL1	3	c15	0	c3	4	32	11.8 CLUSTERECTLR, Cluster Extended Control Register on page 105
CLUSTERPWRCTLR_EL1	3	c15	0	c3	5	32	11.13 CLUSTERPWRCTLR, Cluster Power Control Register on page 119
CLUSTERPWRDN_EL1	3	c15	0	c3	6	32	11.14 CLUSTERPWRDN, Cluster Powerdown Register on page 121
CLUSTERPWRSTAT_EL1	3	c15	0	c3	7	32	11.15 CLUSTERPWRSTAT, Cluster Power Status Register on page 123
CLUSTERTHREADSID_EL1	3	c15	0	c4	0	32	11.18 CLUSTERTHREADSID, Cluster Thread Scheme ID Register on page 129
CLUSTERACPSID_EL1	3	c15	0	c4	1	32	11.4 CLUSTERACPSID, Cluster ACP Scheme ID Register on page 97
CLUSTERSTASHSID_EL1	3	c15	0	c4	2	32	11.17 CLUSTERSTASHSID, Cluster Stash Scheme ID Register on page 127

Register mnemonic	Op0	CRn	Op1	CRm	Op2	Width	Register name and description
CLUSTERPARTCR_EL1	3	c15	0	c4	3	32	11.12 CLUSTERPARTCR, Cluster Partition Control Register on page 114
CLUSTERBUSQOS_EL1	3	c15	0	c4	4	32	11.6 CLUSTERBUSQOS, Cluster Bus QoS Control Register on page 99
CLUSTERL3HIT_EL1	3	c15	0	c4	5	32	11.10 CLUSTERL3HIT, Cluster L3 Hit Counter Register on page 111
CLUSTERL3MISS_EL1	3	c15	0	c4	6	32	11.11 CLUSTERL3MISS, Cluster L3 Miss Counter Register on page 113
CLUSTERTHREADSIDOVR_EL1	3	c15	0	c4	7	32	11.19 CLUSTERTHREADSIDOVR, Cluster Thread Scheme ID Override Register on page 130

11.4 CLUSTERACPSID, Cluster ACP Scheme ID Register

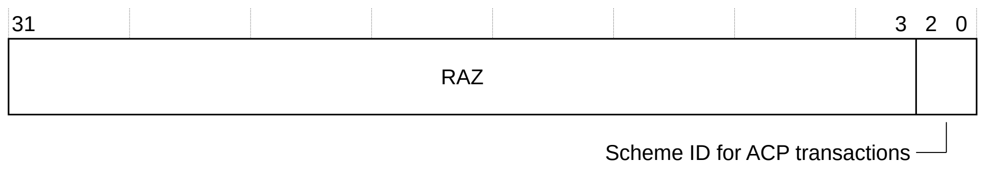
The CLUSTERACPSID register provides the scheme ID for ACP transactions.

Bit field descriptions

This description applies to both the AArch32 (CLUSTERACPSID) and AArch64 (CLUSTERACPSID_EL1) registers.

CLUSTERACPSID is a 32-bit register, and is part of SCU and L3 cache configuration registers.

Figure 11-1: CLUSTERACPSID bit assignments



RAZ, [31:3]

Read-As-Zero.

Scheme ID for ACP transactions, [2:0]

These bits reset to `0b000`.

Configurations

The AArch32 CLUSTERACPSID register is architecturally mapped to the AArch64 CLUSTERACPSID_EL1 register.

Usage constraints

Accessing the CLUSTERACPSID

In AArch64 state (CLUSTERACPSID_EL1):

To read this register in AArch64 state (CLUSTERACPSID_EL1) into a general-purpose register, use the MRS instruction with the following syntax:

```
MRS <Xt>, S3_0_C15_C4_1; Read CLUSTERACPSID_EL1 into Xt
```

To write this register in AArch64 state (CLUSTERACPSID_EL1) from a general-purpose register, use the MSR instruction with the following syntax:

```
MSR S3_0_C15_C4_1, <Xt>; Write Xt into CLUSTERACPSID_EL1
```

In AArch32 state (CLUSTERACPSID):

To read this register in AArch32 state (CLUSTERACPSID) into a general-purpose register, use the MRC (or MRC2) instruction with the following syntax:

```
MRC p15, 0, <Rt>, c15, c4, 1; Read CLUSTERACPSID into Rt
```

To write this register in AArch32 state (CLUSTERACPSID) from a general-purpose register, use the MCR (or MCR2) instruction with the following syntax:

```
MCR p15, 0, <Rt>, c15, c4, 1; Write Rt into CLUSTERACPSID
```

Accessibility

This register is accessible in software as follows:

<systemreg>	Control			Accessibility			
	E2H	TGE	NS	EL0	EL1	EL2	EL3
CLUSTERACPSID	x	x	0	-	RW	n/a	RW
CLUSTERACPSID	x	0	1	-	RW	RW	RW
CLUSTERACPSID	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

This register is write accessible in EL1 if ACTLR_EL3.SMEN is 1 and ACTLR_EL2.SMEN is 1, or ACTLR_EL3.SMEN is 1 and SCR.NS is 0.

If write access is not permitted, then Trap to the lowest Exception level that denied access (EL2 or EL3).

11.5 CLUSTERACTLR, Cluster Auxiliary Control Register

The CLUSTERACTLR register is Reserved.

Traps and enables

This register is write accessible in EL1 if ACTLR_EL3.ACTLREN is 1 and ACTLR_EL2.ACTLREN is 1, or ACTLR_EL3.ACTLREN is 1 and SCR.NS is 0.

If write access is not permitted, then trap to the lowest Exception level that denied access (EL2 or EL3).

11.6 CLUSTERBUSQOS, Cluster Bus QoS Control Register

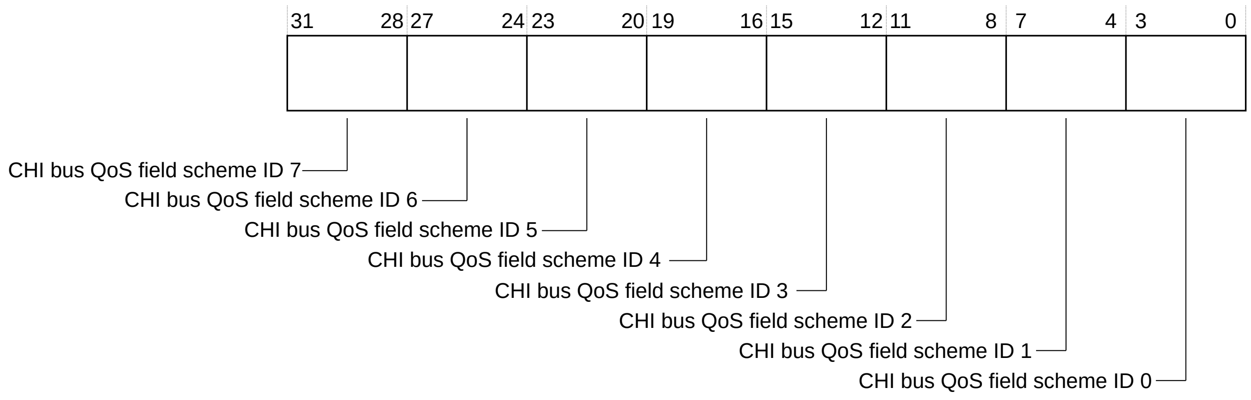
The CLUSTERBUSQOS provides control for the CHI *Quality of Service* (QoS) fields for different scheme IDs.

Bit field descriptions

This description applies to both the AArch32 (CLUSTERBUSQOS) and AArch64 (CLUSTERBUSQOS_EL1) registers.

CLUSTERBUSQOS is a 32-bit register, and is part of SCU and L3 cache configuration registers.

Figure 11-2: CLUSTERBUSQOS bit assignments


31
28
27
24
23
20
19
16
15
12
11
8
7
4
3
0
CHI bus QoS field scheme ID 7
CHI bus QoS field scheme ID 6
CHI bus QoS field scheme ID 5
CHI bus QoS field scheme ID 4
CHI bus QoS field scheme ID 3
CHI bus QoS field scheme ID 2
CHI bus QoS field scheme ID 1
CHI bus QoS field scheme ID 0

CHI bus QoS field scheme ID 7, [31:28]

Value driven on the CHI bus QoS field for scheme ID 7.

These bits reset to `0xE`.

CHI bus QoS field scheme ID 6, [27:24]

Value driven on the CHI bus QoS field for scheme ID 6.

These bits reset to `0xE`.

CHI bus QoS field scheme ID 5, [23:20]

Value driven on the CHI bus QoS field for scheme ID 5.

These bits reset to `0xE`.

CHI bus QoS field scheme ID 4, [19:16]

Value driven on the CHI bus QoS field for scheme ID 4.

These bits reset to `0xE`.

CHI bus QoS field scheme ID 3, [15:12]

Value driven on the CHI bus QoS field for scheme ID 3.

These bits reset to `0xE`.

CHI bus QoS field scheme ID 2, [11:8]

Value driven on the CHI bus QoS field for scheme ID 2.

These bits reset to `0xE`.

CHI bus QoS field scheme ID 1, [7:4]

Value driven on the CHI bus QoS field for scheme ID 1.

These bits reset to `0xE`.

CHI bus QoS field scheme ID 0, [3:0]

Value driven on the CHI bus QoS field for scheme ID 0.

These bits reset to `0xE`.

Configurations

The AArch32 CLUSTERBUSQOS register is architecturally mapped to the AArch64 CLUSTERBUSQOS_EL1 register.

Usage constraints

Accessing the CLUSTERBUSQOS

In AArch64 state (CLUSTERBUSQOS_EL1):

To read this register in AArch64 state (CLUSTERBUSQOS_EL1) into a general-purpose register, use the MRS instruction with the following syntax:

```
MRS <Xt>, S3_0_C15_C4_4; Read CLUSTERBUSQOS_EL1 into Xt
```

To write this register in AArch64 state (CLUSTERBUSQOS_EL1) from a general-purpose register, use the MSR instruction with the following syntax:

```
MSR S3_0_C15_C4_4, <Xt>; Write Xt into CLUSTERBUSQOS_EL1
```

In AArch32 state (CLUSTERBUSQOS):

To read this register in AArch32 state (CLUSTERBUSQOS) into a general-purpose register, use the MRC (or MRC2) instruction with the following syntax:

```
MRC p15, 0, <Rt>, c15, c4, 4; Read CLUSTERBUSQOS into Rt
```

To write this register in AArch32 state (CLUSTERBUSQOS) from a general-purpose register, use the MCR (or MCR2) instruction with the following syntax:

```
MCR p15, 0, <Rt>, c15, c4, 4; Write Rt into CLUSTERBUSQOS
```

Accessibility

This register is accessible in software as follows:

<systemreg>	Control			Accessibility			
	E2H	TGE	NS	EL0	EL1	EL2	EL3
CLUSTERBUSQOS	x	x	0	-	RW	n/a	RW
CLUSTERBUSQOS	x	0	1	-	RW	RW	RW
CLUSTERBUSQOS	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

This register is write accessible in EL1 if ACTLR_EL3.SMEN is 1 and ACTLR_EL2.SMEN is 1, or ACTLR_EL3.SMEN is 1 and SCR.NS is 0.

If write access is not permitted, then trap to the lowest Exception level that denied access (EL2 or EL3).

11.7 CLUSTERCFR, Cluster Configuration Register

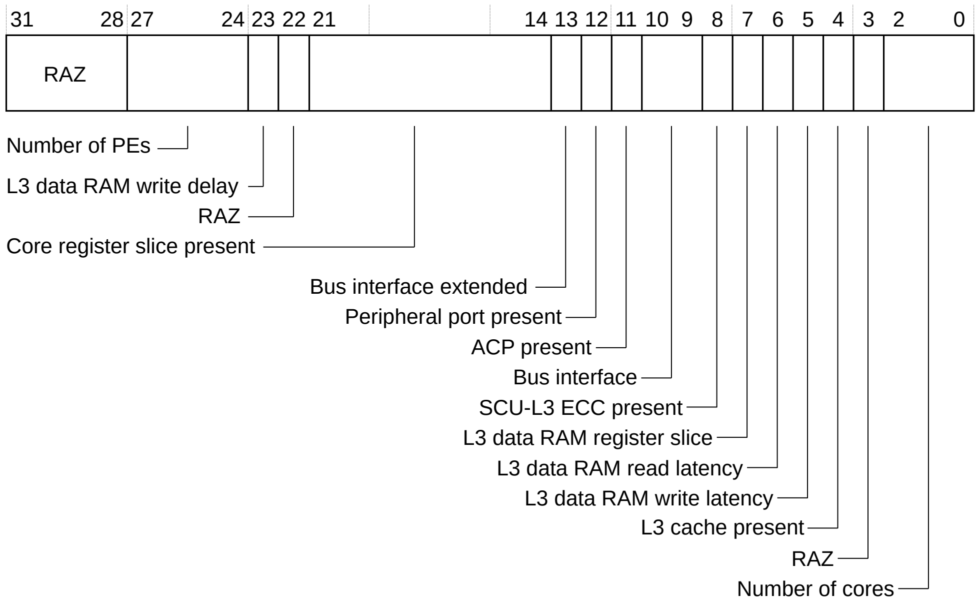
The CLUSTERCFR register contains details of the hardware configuration of the cluster. This register is read-only write-ignores RO (WI), and is common to all execution threads.

Bit field descriptions

This description applies to both the AArch32 (CLUSTERCFR) and AArch64 (CLUSTERCFR_EL1) registers.

CLUSTERCFR is a 32-bit register, and is part of SCU and L3 cache configuration registers.

Figure 11-3: CLUSTERCFR bit assignments



RAZ, [31:28]

Read-As-Zero

Number of PEs, [27:24]

PE - 1, where PE is the number of processing elements in the cluster. Each core contains either one or two PEs.

L3 data RAM write delay, [23]

The possible values are:

0	Writes are not limited.
1	Writes are limited to one write every three cycles.

RAZ, [22]

Read-As-Zero

Core register slice present, [21:14]

Core register slice is present. Each bit represents a core, with bit[14] for core 0 up to bit[21] for core 7:

0	No register slice is present.
1	Register slice is present.

Bus interface extended, [13]

See Bus interface (bits[10:9]).

Peripheral port present, [12]

Peripheral port is present:

`0`	No peripheral port is present.
`1`	Peripheral port is present.

ACP present, [11]

ACP interface is present:

`0`	No ACP interface is present.
`1`	ACP interface is present.

Bus interface, [10:9]

Bus interface configuration:

`0b00`	Single 128-bit ACE.
`0b01`	Dual 128-bit ACE.
`0b10`	Single 128-bit CHI.
`0b11`	If Bus interface extended (bit [13]) is 1, then Dual 256-bit CHI. If Bus interface extended (bit [13]) is 0, then Single 256-bit CHI.

SCU-L3 ECC present, [8]

SCU-L3 is configured with ECC:

`0`	SCU-L3 is not configured with ECC.
`1`	SCU-L3 is configured with ECC.

L3 data RAM register slice, [7]

L3 data RAM read register slice:

`0`	No register slice is present.
`1`	Register slice is present.

L3 data RAM read latency, [6]

L3 data RAM read latency:

`0`	Two-cycle output delay from the L3 data RAMs.
`1`	Three-cycle output delay from L3 data RAMs.

L3 data RAM write latency, [5]

L3 data RAM write latency:

`0`	One cycle input delay from the L3 data RAMs.
`1`	Two cycle input delay from the L3 data RAMs.

L3 cache present, [4]

L3 cache is present:

`0`	No L3 cache is present.
`1`	L3 cache is present.

RAZ, [3]

Read-As-Zero

Number of cores, [2:0]

Number of cores present in the cluster:

`0b000`	One core, `Core0`.
`0b001`	Two cores, `Core0-1`.
`0b010`	Three cores, `Core0-2`.
`0b011`	Four cores, `Core0-3`.
`0b100`	Five cores, `Core0-4`.
`0b101`	Six cores, `Core0-5`.
`0b110`	Seven cores, `Core0-6`.
`0b111`	Eight cores, `Core0-7`.

Configurations

The AArch32 CLUSTERCFR register is architecturally mapped to the AArch64 CLUSTERCFR_EL1 register.

Usage constraints

Accessing the CLUSTERCFR

In AArch64 state (CLUSTERCFR_EL1):

To read this register in AArch64 state (CLUSTERCFR_EL1) into a general-purpose register, use the MRS instruction with the following syntax:

```
MRS <Xt>, S3_0_C15_C3_0; Read CLUSTERCFR_EL1 into Xt
```

To write this register in AArch64 state (CLUSTERCFR_EL1) from a general-purpose register, use the MSR instruction with the following syntax:

```
MSR S3_0_C15_C3_0, <Xt>; Write Xt into CLUSTERCFR_EL1
```

In AArch32 state (CLUSTERCFR):

To read this register in AArch32 state (CLUSTERCFR) into a general-purpose register, use the MRC (or MRC2) instruction with the following syntax:

```
MRC p15, 0, <Rt>, c15, c3, 0; Read CLUSTERCFR into Rt
```

To write this register in AArch32 state (CLUSTERCFR) from a general-purpose register, use the MCR (or MCR2) instruction with the following syntax:

```
MCR p15, 0, <Rt>, c15, c3, 0; Write Rt into CLUSTERCFR
```

Accessibility

This register is accessible in software as follows:

<systemreg>	Control			Accessibility			
	E2H	TGE	NS	EL0	EL1	EL2	EL3
CLUSTERCFR	x	x	0	-	RO/WI	n/a	RO/WI
CLUSTERCFR	x	0	1	-	RO/WI	RO/WI	RO/WI
CLUSTERCFR	x	1	1	-	n/a	RO/WI	RO/WI

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

Traps and enables

There are no special traps and enables.

11.8 CLUSTERECTLR, Cluster Extended Control Register

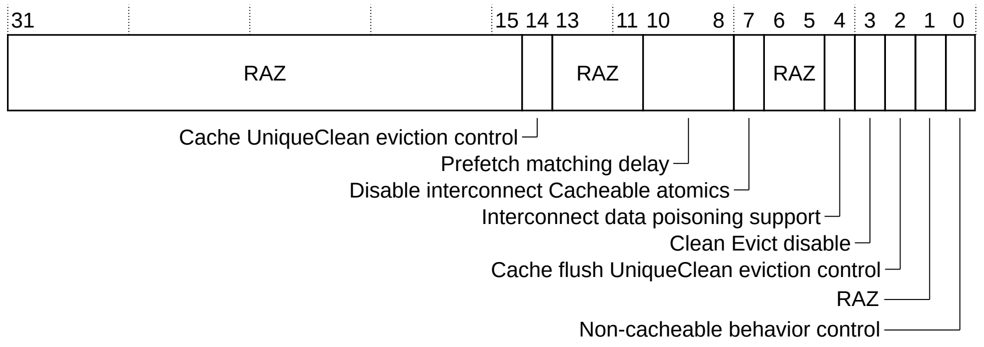
The CLUSTERECTLR register provides implementation-specific control of the microarchitecture. It must only be written to as part of the initial system configuration setup following reset. This register is RW, and is common to all execution threads.

Bit field descriptions

CLUSTERECTLR is a 32-bit register, and is part of SCU and L3 cache configuration registers.

This description applies to both the AArch32 (CLUSTERECTLR) and AArch64 (CLUSTERECTLR_EL1) registers.

Figure 11-4: CLUSTERECTLR bit assignments


31
15 14 13
11 10
8 7 6 5 4 3 2 1 0
RAZ
RAZ
RAZ
Cache UniqueClean eviction control
Prefetch matching delay
Disable interconnect Cacheable atomics
Interconnect data poisoning support
Clean Evict disable
Cache flush UniqueClean eviction control
RAZ
Non-cacheable behavior control

RAZ, [31:15]

Read-As-Zero

Cache UniqueClean eviction control, [14]

Enables sending WriteEvict transactions on the ACE or CHI master for UniqueClean evictions. WriteEvict transactions update downstream caches that are outside the cluster. Enable WriteEvict transactions only if there is an L4 or system cache implemented in the system. The possible values are:

`0`	Disables sending data with UniqueClean evictions.
`1`	Enables sending data with UniqueClean evictions.

For ACE, this bit resets to `0`.

For CHI, this bit resets to `1`.

RAZ, [13:11]

Read-As-Zero

Prefetch matching delay, [10:8]

Prefetch matching delay. Controls the amount of time a prefetch waits for a possible match with a later read. Encoded as powers of 2, from 1-128.

These bits reset to `0x5`.

Disable interconnect Cacheable atomics, [7]

Disable Cacheable atomics being sent to the interconnect. The possible values are:

`0`	If BROADCASTATOMIC is set HIGH, Cacheable atomics are sent to the interconnect.
`1`	Cacheable atomics are handled inside the cluster.

This bit has no effect for ACE configurations.

This bit resets to `0`.

RAZ, [6:5]

Read-As-Zero

Interconnect data poisoning support, [4]

Interconnect data poisoning support. This bit is RAZ for ACE configurations. The possible values are:

`0`	Interconnect does not support data poisoning. Therefore nERRIRQ is asserted when poisoned data is evicted from the cluster or returned to a snoop.
`1`	Interconnect supports data poisoning. Therefore no error recovery interrupt is generated when poisoned data is evicted from the cluster or returned to a snoop.

For ACE, this bit resets to `0`.

For CHI, this bit resets to `1`.

Cache Evict disable, [3]

Disables sending of Evict transactions on the ACE or CHI master for clean cache lines that are evicted from the cluster. Evict transactions are required only if the external interconnect contains a snoop filter that requires notification when the cluster evicts the cache line. The possible values are:

`0`	Enables sending Evict transactions.
`1`	Disables sending Evict transactions.

This bit resets to `0`.

Cache flush UniqueClean eviction control, [2]

Disables the sending of WriteEvict requests on the ACE or CHI master when powering down part or all of the L3 cache. The possible values are:

`0`	Evictions during L3 cache powerdown behave like normal evictions. This is the reset value.
`1`	Disables sending data with UniqueClean evictions caused by powering down the L3 cache.

RAZ, [1]

Read-As-Zero

Non-cacheable behavior control, [0]

Enable Normal Non-cacheable writes to all master interfaces and, when ACE is configured, also disable the limit on the number of Normal Non-cacheable writes. The possible values are:

0	All Normal Non-cacheable and Device transactions are sent to interface 0. When ACE is configured, Device and Normal Non-cacheable writes are limited to 15 outstanding transactions.
1	If dual master interfaces are configured, Normal Non-cacheable transactions are interleaved between master interfaces like Cacheable transactions. Device transactions are still sent to interface 0. When ACE is configured, only Device writes are limited to 15 outstanding transactions. The limit is removed for Normal Non-cacheable transactions.

If this bit is set to 1, then further writes to this bit are ignored.

Note

Setting this bit might have implications on the behavior of system components, for example a CPE-425 Coherent PCIe Extension in an ACE system.

Configurations

The AArch32 CLUSTERECTLR register is architecturally mapped to the AArch64 CLUSTERECTLR_EL1 register.

Usage constraints

Accessing the CLUSTERECTLR

In AArch64 state (CLUSTERECTLR_EL1):

To read this register in AArch64 state (CLUSTERECTLR_EL1) into a general-purpose register, use the MRS instruction with the following syntax:

```
MRS <Xt>, S3_0_C15_C3_4; Read CLUSTERECTLR_EL1 into Xt
```

To write this register in AArch64 state (CLUSTERECTLR_EL1) from a general-purpose register, use the MSR instruction with the following syntax:

```
MSR S3_0_C15_C3_4, <Xt>; Write Xt into CLUSTERECTLR_EL1
```

In AArch32 state (CLUSTERECTLR):

To read this register in AArch32 state (CLUSTERECTLR) into a general-purpose register, use the MRC (or MRC2) instruction with the following syntax:

```
MRC p15, 0, <Rt>, c15, c3, 4; Read CLUSTERECTLR into Rt
```

To write this register in AArch32 state (CLUSTERECTLR) from a general-purpose register, use the MCR (or MCR2) instruction with the following syntax:

```
MCR p15, 0, <Rt>, c15, c3, 4; Write Rt into CLUSTERECTLR
```

Accessibility

This register is accessible in software as follows:

<systemreg>	Control			Accessibility			
	E2H	TGE	NS	EL0	EL1	EL2	EL3
CLUSTERECTLR	x	x	0	-	RW	n/a	RW
CLUSTERECTLR	x	0	1	-	RW	RW	RW
CLUSTERECTLR	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

This register is write accessible in EL1 if ACTLR_EL3.ECTLREN is 1 and ACTLR_EL2.ECTLREN is 1, or ACTLR_EL3.ECTLR_EN is 1 and SCR.NS is 0.

If write access is not possible, then Trap to the lowest Exception level that denied access (EL2 or EL3).

11.9 CLUSTERIDR, Cluster Main Revision ID Register

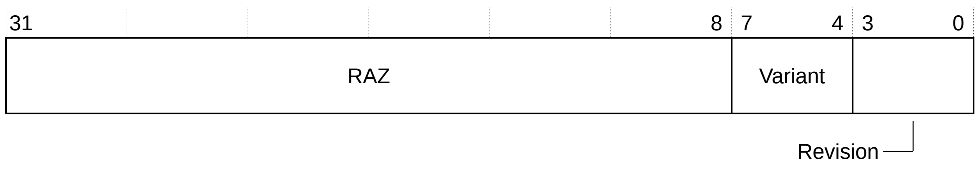
The CLUSTERIDR register contains the revision and patch level of the DSU. This register is read-only write-ignores RO (WI), and is common to all execution threads.

Bit field descriptions

This description applies to both the AArch32 (CLUSTERIDR) and AArch64 (CLUSTERIDR_EL1) registers.

CLUSTERIDR is a 32-bit register, and is part of SCU and L3 cache configuration registers.

Figure 11-5: CLUSTERIDR bit assignments



RAZ, [31:8]

Read-As-Zero.

Variant, [7:4]

Indicates the variant of the DSU. 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:

0x4	r4

Revision, [3:0]

Indicates the minor revision number of the DSU. This is the minor revision number *y* in the p*y* part of the r*x*p*y* description of the product revision status. This value is:

0x1	p1

Configurations

The AArch32 CLUSTERIDR register is architecturally mapped to the AArch64 CLUSTERIDR_EL1 register.

Usage constraints

Accessing the CLUSTERIDR

In AArch64 state (CLUSTERIDR_EL1):

To read this register in AArch64 state (CLUSTERIDR_EL1) into a general-purpose register, use the MRS instruction with the following syntax:

```
MRS <Xt>, S3_0_C15_C3_1; Read CLUSTERIDR_EL1 into Xt
```

To write this register in AArch64 state (CLUSTERIDR_EL1) from a general-purpose register, use the MSR instruction with the following syntax:

```
MSR S3_0_C15_C3_1, <Xt>; Write Xt into CLUSTERIDR_EL1
```

In AArch32 state (CLUSTERIDR):

To read this register in AArch32 state (CLUSTERIDR) into a general-purpose register, use the MRC (or MRC2) instruction with the following syntax:

```
MRC p15, 0, <Rt>, c15, c3, 1; Read CLUSTERIDR into Rt
```

To write this register in AArch32 state (CLUSTERIDR) from a general-purpose register, use the MCR (or MCR2) instruction with the following syntax:

```
MCR p15, 0, <Rt>, c15, c3, 1; Write Rt into CLUSTERIDR
```

Accessibility

This register is accessible in software as follows:

<systemreg>	Control			Accessibility			
	E2H	TGE	NS	EL0	EL1	EL2	EL3
CLUSTERIDR	x	x	0	-	RO/WI	n/a	RO/WI
CLUSTERIDR	x	0	1	-	RO/WI	RO/WI	RO/WI
CLUSTERIDR	x	1	1	-	n/a	RO/WI	RO/WI

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

Traps and enables

There are no traps and enables that affect this register.

11.10 CLUSTERL3HIT, Cluster L3 Hit Counter Register

The CLUSTERL3HIT register contains a count of the L3 cache hits. This register is intended for use in algorithms for determining when to power up or power down portions. This register is RW, and is common to all execution threads.

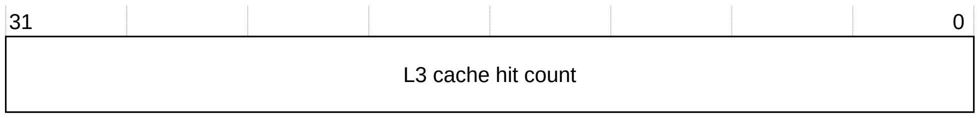
Bit field descriptions

This description applies to both the AArch32 (CLUSTERL3HIT) and AArch64 (CLUSTERL3HIT_EL1) registers.

If CLUSTERPMMDCR.SPME == 0, this counter does not trigger for secure transactions.

CLUSTERL3HIT is a 32-bit register, and is part of SCU and L3 cache configuration registers.

Figure 11-6: CLUSTERL3HIT bit assignments



L3 cache hit count, [31:0]

Count of number of L3 cache hits, for use in portion control calculations. The counter saturates to its maximum value on overflow.

On a write, CLUSTERL3HIT gets set to the specified value.

These bits reset to `0x00000000`.

Configurations

The AArch32 CLUSTERL3HIT register is architecturally mapped to the AArch64 CLUSTERL3HIT_EL1 register.

Usage constraints

Accessing the CLUSTERL3HIT

In AArch64 state (CLUSTERL3HIT_EL1):

To read this register in AArch64 state (CLUSTERL3HIT_EL1) into a general-purpose register, use the MRS instruction with the following syntax:

```
MRS <Xt>, S3_0_C15_C4_5; Read CLUSTERL3HIT_EL1 into Xt
```

To write this register in AArch64 state (CLUSTERL3HIT_EL1) from a general-purpose register, use the MSR instruction with the following syntax:

```
MSR S3_0_C15_C4_5, <Xt>; Write Xt into CLUSTERL3HIT_EL1
```

In AArch32 state (CLUSTERL3HIT):

To read this register in AArch32 state (CLUSTERL3HIT) into a general-purpose register, use the MRC (or MRC2) instruction with the following syntax:

```
MRC p15, 0, <Rt>, c15, c4, 5; Read CLUSTERL3HIT into Rt
```

To write this register in AArch32 state (CLUSTERL3HIT) from a general-purpose register, use the MCR (or MCR2) instruction with the following syntax:

```
MCR p15, 0, <Rt>, c15, c4, 5; Write Rt into CLUSTERL3HIT
```

Accessibility

This register is accessible in software as follows:

<systemreg>	Control			Accessibility			
	E2H	TGE	NS	EL0	EL1	EL2	EL3
CLUSTERL3HIT	x	x	0	-	RW	n/a	RW
CLUSTERL3HIT	x	0	1	-	RW	RW	RW
CLUSTERL3HIT	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

This register is write accessible in EL1 if ACTLR_EL3.PWREN is 1 and ACTLR_EL2.PWREN is 1, or ACTLR_EL3.PWREN is 1 and SCR.NS is 0.

If write access is not possible, then Trap to the lowest Exception level that denied access (EL2 or EL3).

11.11 CLUSTERL3MISS, Cluster L3 Miss Counter Register

The CLUSTERL3MISS register contains a count of the L3 cache misses. This register is intended for use in algorithms for determining when to power up or power down portions. This register is RW, and is common to all execution threads.

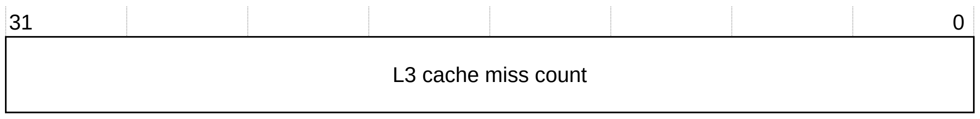
Bit field descriptions

This description applies to both the AArch32 (CLUSTERL3MISS) and AArch64 (CLUSTERL3MISS_EL1) registers.

If CLUSTERPMMDCR.SPME == 0, this counter does not trigger for secure transactions.

CLUSTERL3MISS is a 32-bit register, and is part of SCU and L3 cache configuration registers.

Figure 11-7: CLUSTERL3MISS bit assignments



L3 cache miss count, [31:0]

Count of number of L3 cache misses, for use in portion control calculations. The counter saturates to its maximum value on overflow.

On a write, CLUSTERL3MISS gets set to the specified value.

These bits reset to `0x00000000`.

Configurations

The AArch32 CLUSTERL3MISS register is architecturally mapped to the AArch64 CLUSTERL3MISS_EL1 register.

Usage constraints

Accessing the CLUSTERL3MISS

In AArch64 state (CLUSTERL3MISS_EL1):

To read this register in AArch64 state (CLUSTERL3MISS_EL1) into a general-purpose register, use the MRS instruction with the following syntax:

```
MRS <Xt>, S3_0_C15_C4_6; Read CLUSTERL3MISS_EL1 into Xt
```

To write this register in AArch64 state (CLUSTERL3MISS_EL1) from a general-purpose register, use the MSR instruction with the following syntax:

```
MSR S3_0_C15_C4_6, <Xt>; Write Xt into CLUSTERL3MISS_EL1
```

In AArch32 state (CLUSTERL3MISS):

To read this register in AArch32 state (CLUSTERL3MISS) into a general-purpose register, use the MRC (or MRC2) instruction with the following syntax:

```
MRC p15, 0, <Rt>, c15, c4, 6; Read CLUSTERL3MISS into Rt
```

To write this register in AArch32 state (CLUSTERL3MISS) from a general-purpose register, use the MCR (or MCR2) instruction with the following syntax:

```
MCR p15, 0, <Rt>, c15, c4, 6; Write Rt into CLUSTERL3MISS
```

Accessibility

This register is accessible in software as follows:

<systemreg>	Control			Accessibility			
	E2H	TGE	NS	EL0	EL1	EL2	EL3
CLUSTERL3MISS	x	x	0	-	RW	n/a	RW
CLUSTERL3MISS	x	0	1	-	RW	RW	RW
CLUSTERL3MISS	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

This register is write accessible in EL1 if ACTLR_EL3.PWREN is 1 and ACTLR_EL2.PWREN is 1, or ACTLR_EL3.PWREN is 1 and SCR.NS is 0.

If write access is not possible, then Trap to the lowest Exception level that denied access (EL2 or EL3).

11.12 CLUSTERPARTCR, Cluster Partition Control Register

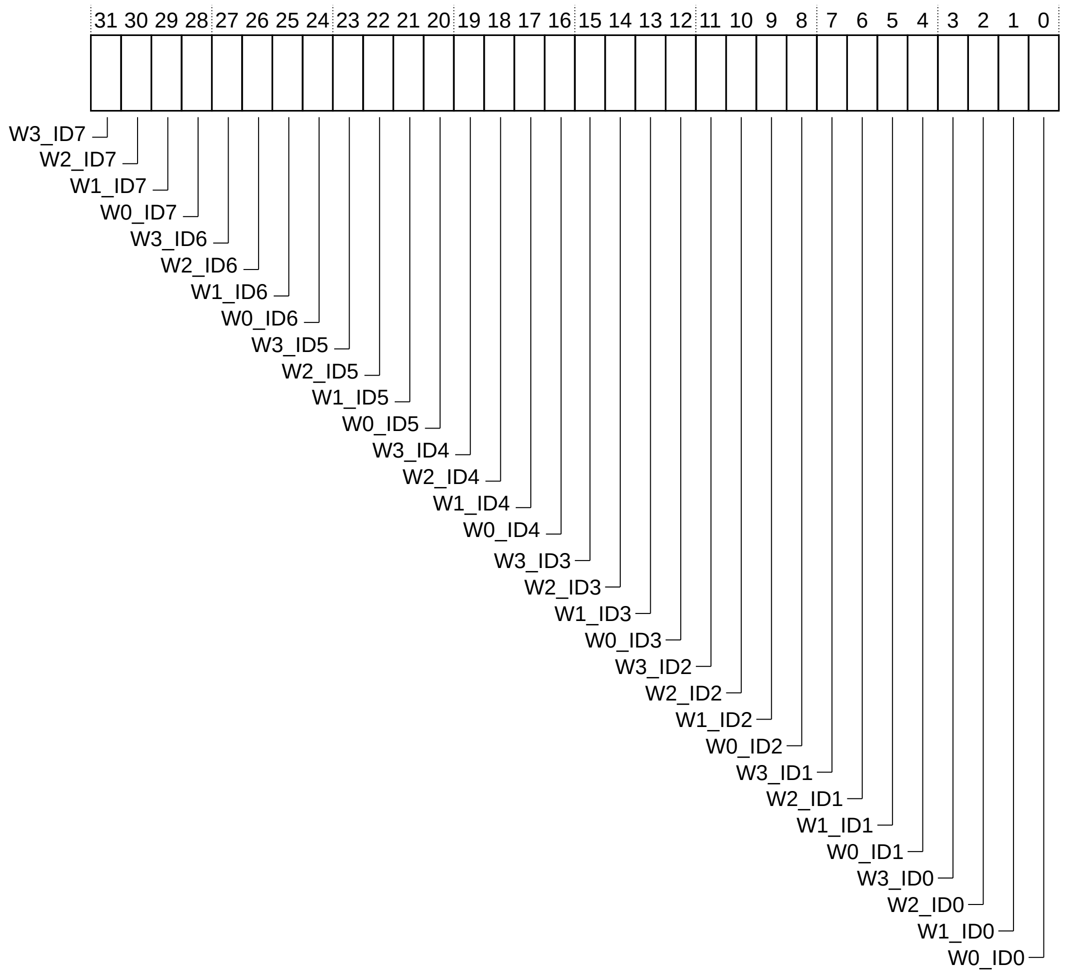
The CLUSTERPARTCR register controls a group of ways to be marked as private to a scheme ID. This register is RW.

Bit field descriptions

This description applies to both the AArch32 (CLUSTERPARTCR) and AArch64 (CLUSTERPARTCR_EL1) registers.

CLUSTERPARTCR is a 32-bit register, and is part of SCU and L3 cache configuration registers.

Figure 11-8: CLUSTERPARTCR bit assignments



Each bit, if set, indicates that a group of four ways is allocated as private to that scheme ID. If more than one scheme ID assigns the same group of ways as private, then those ways are shared between the scheme IDs that have assigned them as private. All ways not assigned to any scheme

ID are treated as shared between all scheme IDs. If a scheme ID does not have any private ways allocated, and there are no remaining shared ways, then any use of the scheme ID will allocate to way group 0, as this is considered a programming error.

Way group 3 is assigned as private to scheme ID 7, [31]

This bit resets to `0`.

Way group 2 is assigned as private to scheme ID 7, [30]

This bit resets to `0`.

Way group 1 is assigned as private to scheme ID 7, [29]

This bit resets to `0`.

Way group 0 is assigned as private to scheme ID 7, [28]

This bit resets to `0`.

Way group 3 is assigned as private to scheme ID 6, [27]

This bit resets to `0`.

Way group 2 is assigned as private to scheme ID 6, [26]

This bit resets to `0`.

Way group 1 is assigned as private to scheme ID 6, [25]

This bit resets to `0`.

Way group 0 is assigned as private to scheme ID 6, [24]

This bit resets to `0`.

Way group 3 is assigned as private to scheme ID 5, [23]

This bit resets to `0`.

Way group 2 is assigned as private to scheme ID 5, [22]

This bit resets to `0`.

Way group 1 is assigned as private to scheme ID 5, [21]

This bit resets to `0`.

Way group 0 is assigned as private to scheme ID 5, [20]

This bit resets to `0`.

Way group 3 is assigned as private to scheme ID 4, [19]

This bit resets to `0`.

Way group 2 is assigned as private to scheme ID 4, [18]

This bit resets to `0`.

Way group 1 is assigned as private to scheme ID 4, [17]

This bit resets to `0`.

Way group 0 is assigned as private to scheme ID 4, [16]

This bit resets to `0`.

Way group 3 is assigned as private to scheme ID 3, [15]

This bit resets to `0`.

Way group 2 is assigned as private to scheme ID 3, [14]

This bit resets to `0`.

Way group 1 is assigned as private to scheme ID 3, [13]

This bit resets to `0`.

Way group 0 is assigned as private to scheme ID 3, [12]

This bit resets to `0`.

Way group 3 is assigned as private to scheme ID 2, [11]

This bit resets to `0`.

Way group 2 is assigned as private to scheme ID 2, [10]

This bit resets to `0`.

Way group 1 is assigned as private to scheme ID 2, [9]

This bit resets to `0`.

Way group 0 is assigned as private to scheme ID 2, [8]

This bit resets to `0`.

Way group 3 is assigned as private to scheme ID 1, [7]

This bit resets to `0`.

Way group 2 is assigned as private to scheme ID 1, [6]

This bit resets to `0`.

Way group 1 is assigned as private to scheme ID 1, [5]

This bit resets to `0`.

Way group 0 is assigned as private to scheme ID 1, [4]

This bit resets to `0`.

Way group 3 is assigned as private to scheme ID 0, [3]

This bit resets to `0`.

Way group 2 is assigned as private to scheme ID 0, [2]

This bit resets to `0`.

Way group 1 is assigned as private to scheme ID 0, [1]

This bit resets to `0`.

Way group 0 is assigned as private to scheme ID 0, [0]

This bit resets to `0`.

Configurations

The AArch32 CLUSTERPARTCR register is architecturally mapped to the AArch64 CLUSTERPARTCR_EL1 register.

Usage constraints

Accessing the CLUSTERPARTCR

In AArch64 state (CLUSTERPARTCR_EL1):

To read this register in AArch64 state (CLUSTERPARTCR_EL1) into a general-purpose register, use the MRS instruction with the following syntax:

```
MRS <Xt>, S3_0_C15_C4_3; Read CLUSTERPARTCR_EL1 into Xt
```

To write this register in AArch64 state (CLUSTERPARTCR_EL1) from a general-purpose register, use the MSR instruction with the following syntax:

```
MSR S3_0_C15_C4_3, <Xt>; Write Xt into CLUSTERPARTCR_EL1
```

In AArch32 state (CLUSTERPARTCR):

To read this register in AArch32 state (CLUSTERPARTCR) into a general-purpose register, use the MRC (or MRC2) instruction with the following syntax:

```
MRC p15, 0, <Rt>, c15, c4, 3; Read CLUSTERPARTCR into Rt
```

To write this register in AArch32 state (CLUSTERPARTCR) from a general-purpose register, use the MCR (or MCR2) instruction with the following syntax:

```
MCR p15, 0, <Rt>, c15, c4, 3; Write Rt into CLUSTERPARTCR
```

Accessibility

This register is accessible in software as follows:

<systemreg>	Control			Accessibility			
	E2H	TGE	NS	EL0	EL1	EL2	EL3
CLUSTERPARTCR	x	x	0	-	RW	n/a	RW
CLUSTERPARTCR	x	0	1	-	RW	RW	RW
CLUSTERPARTCR	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

This register is write accessible in EL1 if ACTLR_EL3.SMEN is 1 and ACTLR_EL2.SMEN is 1, or ACTLR_EL3.SMEN is 1 and SCR.NS is 0.

If write access is not permitted, then Trap to the lowest Exception level that denied access (EL2 or EL3).

11.13 CLUSTERPWRCTLR, Cluster Power Control Register

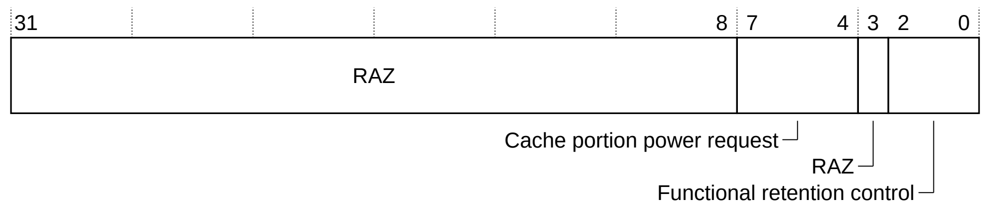
The CLUSTERPWRCTLR register controls power features of the cluster. This register is RW, and is common to all execution threads.

Bit field descriptions

This description applies to both the AArch32 (CLUSTERPWRCTLR) and AArch64 (CLUSTERPWRCTLR_EL1) registers.

CLUSTERPWRCTLR is a 32-bit register, and is part of SCU and L3 cache configuration registers.

Figure 11-9: CLUSTERPWRCTLR bit assignments



RAZ, [31:8]

Read-As-Zero.

Cache portion power request, [7:4]

These bits are output on CLUSTERPACTIVE[19:16] to indicate to the power controller which cache portions must be powered. This is an advisory status to the power controller, and does not cause the P-Channel to deny any requests that are based on this status.

The reset values of these bits depend on the cluster P-Channel initialization state.

RAZ, [3]

Read-As-Zero.

Functional retention control, [2:0]

Duration of inactivity before the DSU uses CLUSTERPACTIVE to request functional retention. The possible values of these bits are:

`0b000`	Disable the retention circuit. This is the default condition.
`0b001`	Two Architectural Timer ticks are required before retention entry.
`0b010`	Eight Architectural Timer ticks are required before retention entry.
`0b011`	32 Architectural Timer ticks are required before retention entry.

0b100	64 Architectural Timer ticks are required before retention entry.
0b101	128 Architectural Timer ticks are required before retention entry.
0b110	256 Architectural Timer ticks are required before retention entry.
0b111	512 Architectural Timer ticks are required before retention entry.

These bits reset to `0b000`.

Configurations

The AArch32 CLUSTERPWRCTLR register is architecturally mapped to the AArch64 CLUSTERPWRCTLR_EL1 register.

Usage constraints

Accessing the CLUSTERPWRCTLR

In AArch64 state (CLUSTERPWRCTLR_EL1):

To read this register in AArch64 state (CLUSTERPWRCTLR_EL1) into a general-purpose register, use the MRS instruction with the following syntax:

```
MRS <Xt>, S3_0_C15_C3_5; Read CLUSTERPWRCTLR_EL1 into Xt
```

To write this register in AArch64 state (CLUSTERPWRCTLR_EL1) from a general-purpose register, use the MSR instruction with the following syntax:

```
MSR S3_0_C15_C3_5, <Xt>; Write Xt into CLUSTERPWRCTLR_EL1
```

In AArch32 state (CLUSTERPWRCTLR):

To read this register in AArch32 state (CLUSTERPWRCTLR) into a general-purpose register, use the MRC (or MRC2) instruction with the following syntax:

```
MRC p15, 0, <Rt>, c15, c3, 5; Read CLUSTERPWRCTLR into Rt
```

To write this register in AArch32 state (CLUSTERPWRCTLR) from a general-purpose register, use the MCR (or MCR2) instruction with the following syntax:

```
MCR p15, 0, <Rt>, c15, c3, 5; Write Rt into CLUSTERPWRCTLR
```

Accessibility

This register is accessible in software as follows:

<systemreg>	Control			Accessibility			
	E2H	TGE	NS	EL0	EL1	EL2	EL3
CLUSTERPWRCTLR	x	x	0	-	RW	n/a	RW
CLUSTERPWRCTLR	x	0	1	-	RW	RW	RW
CLUSTERPWRCTLR	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

This register is write accessible in EL1 if ACTLR_EL3.PWREN is 1 and ACTLR_EL2.PWREN is 1, or ACTLR_EL3.PWREN is 1 and SCR.NS is 0.

If write access is not possible, then Trap to the lowest Exception level that denied access (EL2 or EL3).

11.14 CLUSTERPWRDN, Cluster Powerdown Register

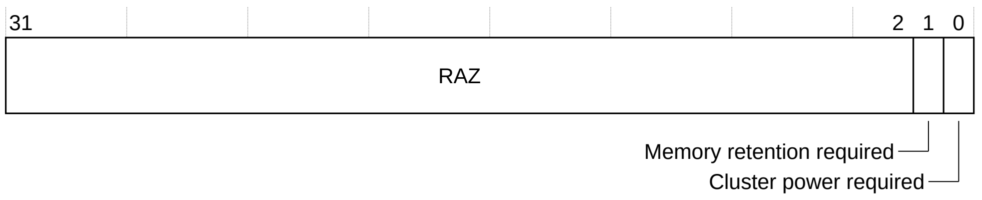
The CLUSTERPWRDN register controls powerdown requirements of the cluster. This register is RW, and is banked for each thread of execution.

Bit field descriptions

This description applies to both the AArch32 (CLUSTERPWRDN) and AArch64 (CLUSTERPWRDN_EL1) registers.

CLUSTERPWRDN is a 32-bit register, and is part of SCU and L3 cache configuration registers.

Figure 11-10: CLUSTERPWRDN bit assignments



RAZ, [31:2]

Read-As-Zero.

Memory retention required, [1]

The possible values are:

0	Indicates on CLUSTERPACTIVE that memory retention is not required when all cores are powered down.
1	Indicates on CLUSTERPACTIVE that memory retention is required when all cores are powered down.

This signal provides an advisory status to the power controller. It does not deny a P-Channel request to powerdown the cluster.

This bit resets to `0`.

Cluster power required, [0]

The possible values are:

`0`	Indicates on CLUSTERPACTIVE that cluster power is not required when all cores are powered down.
`1`	Indicates on CLUSTERPACTIVE that cluster power is required even when all cores are powered down.

This signal provides an advisory status to the power controller. It does not deny a P-Channel request to powerdown the cluster.

This bit resets to `0`.

Configurations

The AArch32 CLUSTERPWRDN register is architecturally mapped to the AArch64 CLUSTERPWRDN_EL1 register.

Usage constraints

Accessing the CLUSTERPWRDN

In AArch64 state (CLUSTERPWRDN_EL1):

To read this register in AArch64 state (CLUSTERPWRDN_EL1) into a general-purpose register, use the MRS instruction with the following syntax:

```
MRS <Xt>, S3_0_C15_C3_6; Read CLUSTERPWRDN_EL1 into Xt
```

To write this register in AArch64 state (CLUSTERPWRDN_EL1) from a general-purpose register, use the MSR instruction with the following syntax:

```
MSR S3_0_C15_C3_6, <Xt>; Write Xt into CLUSTERPWRDN_EL1
```

In AArch32 state (CLUSTERPWRDN):

To read this register in AArch32 state (CLUSTERPWRDN) into a general-purpose register, use the MRC (or MRC2) instruction with the following syntax:

```
MRC p15, 0, <Rt>, c15, c3, 6; Read CLUSTERPWRDN into Rt
```

To write this register in AArch32 state (CLUSTERPWRDN) from a general-purpose register, use the MCR (or MCR2) instruction with the following syntax:

```
MCR p15, 0, <Rt>, c15, c3, 6; Write Rt into CLUSTERPWRDN
```

Accessibility

This register is accessible in software as follows:

<systemreg>	Control			Accessibility			
	E2H	TGE	NS	EL0	EL1	EL2	EL3
CLUSTERPWRDN	x	x	0	-	RW	n/a	RW
CLUSTERPWRDN	x	0	1	-	RW	RW	RW
CLUSTERPWRDN	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

This register is write accessible in EL1 if ACTLR_EL3.PWREN is 1 and ACTLR_EL2.PWREN is 1, or ACTLR_EL3.PWREN is 1 and SCR.NS is 0. If write access is not possible, then Trap to the lowest Exception level that denied access (EL2 or EL3).

11.15 CLUSTERPWRSTAT, Cluster Power Status Register

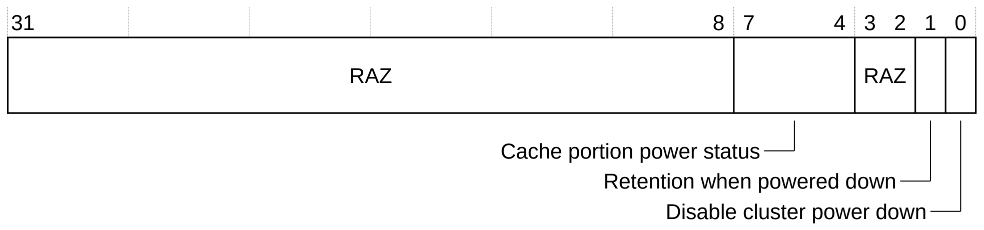
The CLUSTERPWRSTAT register contains the status of the power features. This register is read-only write-ignores RO (WI), and is common to all execution threads.

Bit field descriptions

This description applies to both the AArch32 (CLUSTERPWRSTAT) and AArch64 (CLUSTERPWRSTAT_EL1) registers.

CLUSTERPWRSTAT is a 32-bit register, and is part of SCU and L3 cache configuration registers.

Figure 11-11: CLUSTERPWRSTAT bit assignments


31
8
7
4
3
2
1
0
RAZ
RAZ
Cache portion power status
Retention when powered down
Disable cluster power down

RAZ, [31:8]

Read-As-Zero.

Cache portion power status, [7:4]

These bits indicate which cache portions are currently powered up and available. These bits can be used to determine when the state requested in bits [7:4] of the CLUSTERPWRCTLR has taken effect. The possible values are:

`0b1111`	Ways 0-15 are powered up.

`0b0111`	Ways 0-11 are powered up.
`0b0011`	Ways 0-7 are powered up.
`0b0001`	Ways 0-3 are powered up.
`0b0000`	No ways are powered up.

The reset value for these bits depends on the P-Channel initialization state.

RAZ, [3:2]

Read-As-Zero.

Retention when powered down, [1]

Enabled memory retention when all cores are powered down. This bit is a combined version of all banked bits for each execution thread from the CLUSTERPWRDN register. The possible values are:

`0`	Disables memory retention when all cores are powered down.
`1`	Enables memory retention when all cores are powered down.

This bit resets to `0`.

Disable cluster powerdown, [0]

Disabled cluster powerdown when all cores are powered down. This bit is a combined version of all banked per-thread bits from the CLUSTERPWRDN register. The possible values are:

`0`	Enables cluster powerdown when all cores are powered down.
`1`	Disables cluster powerdown when all cores are powered down.

This bit resets to `0`.

Configurations

The AArch32 CLUSTERPWRSTAT register is architecturally mapped to the AArch64 CLUSTERPWRSTAT_EL1 register.

Usage constraints

Accessing the CLUSTERPWRSTAT

In AArch64 state (CLUSTERPWRSTAT_EL1):

To read this register in AArch64 state (CLUSTERPWRSTAT_EL1) into a general-purpose register, use the MRS instruction with the following syntax:

```
MRS <Xt>, S3_0_C15_C3_7; Read CLUSTERPWRSTAT_EL1 into Xt
```

To write this register in AArch64 state (CLUSTERPWRSTAT_EL1) from a general-purpose register, use the MSR instruction with the following syntax:

```
MSR S3_0_C15_C3_7, <Xt>; Write Xt into CLUSTERPWRSTAT_EL1
```

In AArch32 state (CLUSTERPWRSTAT):

To read this register in AArch32 state (CLUSTERPWRSTAT) into a general-purpose register, use the MRC (or MRC2) instruction with the following syntax:

```
MRC p15, 0, <Rt>, c15, c3, 7; Read CLUSTERPWRSTAT into Rt
```

To write this register in AArch32 state (CLUSTERPWRSTAT) from a general-purpose register, use the MCR (or MCR2) instruction with the following syntax:

```
MCR p15, 0, <Rt>, c15, c3, 7; Write Rt into CLUSTERPWRSTAT
```

Accessibility

This register is accessible in software as follows:

<systemreg>	Control			Accessibility			
	E2H	TGE	NS	EL0	EL1	EL2	EL3
CLUSTERPWRSTAT	x	x	0	-	RO/WI	n/a	RO/WI
CLUSTERPWRSTAT	x	0	1	-	RO/WI	RO/WI	RO/WI
CLUSTERPWRSTAT	x	1	1	-	n/a	RO/WI	RO/WI

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

Traps and enables

This register is write accessible in EL1 if ACTLR_EL3.PWREN is 1 and ACTLR_EL2.PWREN is 1, or ACTLR_EL3.PWREN is 1 and SCR.NS is 0. If write access is not possible, then Trap to the lowest Exception level that denied access (EL2 or EL3).

11.16 CLUSTERREVIDR, Cluster Revision ID Register

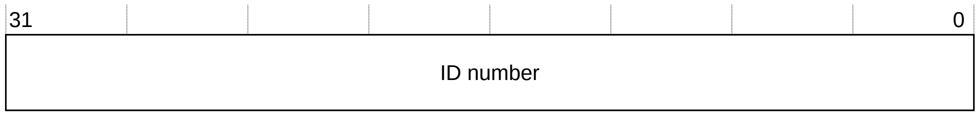
The CLUSTERREVIDR register enables ECO patches applied to the cluster level to be identified by the operating system. This register is read-only write-ignores RO (WI), and is common to all execution threads.

Bit field descriptions

This description applies to both the AArch32 (CLUSTERREVIDR) and AArch64 (CLUSTERREVIDR_EL1) registers.

CLUSTERREVIDR is a 32-bit register, and is part of SCU and L3 cache configuration registers.

Figure 11-12: CLUSTERREVIDR bit assignments



ID number, [31:0]

Implementation-specific revision information. The reset value is determined by the specific DSU implementation.

`0x00000000`	Revision code is zero.

Configurations

The AArch32 CLUSTERREVIDR register is architecturally mapped to the AArch64 CLUSTERREVIDR_EL1 register.

Usage constraints

Accessing the CLUSTERREVIDR

In AArch64 state (CLUSTERREVIDR_EL1):

To read this register in AArch64 state (CLUSTERREVIDR_EL1) into a general-purpose register, use the MRS instruction with the following syntax:

```
MRS <Xt>, S3_0_C15_C3_2; Read CLUSTERREVIDR_EL1 into Xt
```

To write this register in AArch64 state (CLUSTERREVIDR_EL1) from a general-purpose register, use the MSR instruction with the following syntax:

```
MSR S3_0_C15_C3_2, <Xt>; Write Xt into CLUSTERREVIDR_EL1
```

In AArch32 state (CLUSTERREVIDR):

To read this register in AArch32 state (CLUSTERREVIDR) into a general-purpose register, use the MRC (or MRC2) instruction with the following syntax:

```
MRC p15, 0, <Rt>, c15, c3, 2; Read CLUSTERREVIDR into Rt
```

To write this register in AArch32 state (CLUSTERREVIDR) from a general-purpose register, use the MCR (or MCR2) instruction with the following syntax:

```
MCR p15, 0, <Rt>, c15, c3, 2; Write Rt into CLUSTERREVIDR
```

Accessibility

This register is accessible in software as follows:

<systemreg>	Control			Accessibility			
	E2H	TGE	NS	EL0	EL1	EL2	EL3
CLUSTERREVIDR	x	x	0	-	RO/WI	n/a	RO/WI
CLUSTERREVIDR	x	0	1	-	RO/WI	RO/WI	RO/WI
CLUSTERREVIDR	x	1	1	-	n/a	RO/WI	RO/WI

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

Traps and enables

There are no traps and enables affecting this register.

11.17 CLUSTERSTASHSID, Cluster Stash Scheme ID Register

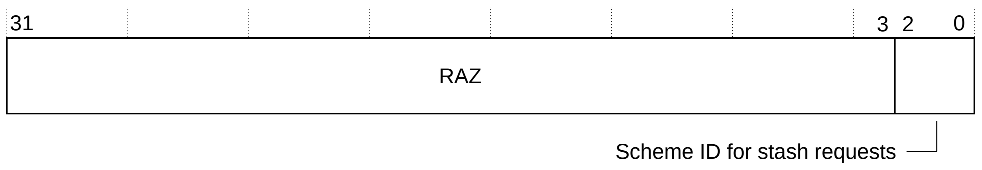
The CLUSTERSTASHSID register provides the scheme ID for stash requests received from the interconnect.

Bit field descriptions

This description applies to both the AArch32 (CLUSTERSTASHID) and AArch64 (CLUSTERSTASHID_EL1) registers.

CLUSTERSTASHSID is a 32-bit register, and is part of SCU and L3 cache configuration registers.

Figure 11-13: CLUSTERSTASHSID bit assignments



RAZ, [31:3]

Read-As-Zero.

Scheme ID for stash requests, [2:0]

Scheme ID for stash requests received from the interconnect.

These bits reset to `0b000`.

Configurations

The AArch32 CLUSTERSTASHSID is architecturally mapped to the AArch64 CLUSTERSTASHSID_EL1 register.

Usage constraints

Accessing the CLUSTERSTASHSID

In AArch64 state (CLUSTERSTASHSID_EL1):

To read this register in AArch64 state (CLUSTERSTASHSID_EL1) into a general-purpose register, use the MRS instruction with the following syntax:

```
MRS <Xt>, S3_0_C15_C4_2; Read CLUSTERSTASHSID_EL1 into Xt
```

To write this register in AArch64 state (CLUSTERSTASHSID_EL1) from a general-purpose register, use the MSR instruction with the following syntax:

```
MSR S3_0_C15_C4_2, <Xt>; Write Xt into CLUSTERSTASHSID_EL1
```

In AArch32 state (CLUSTERSTASHSID):

To read this register in AArch32 state (CLUSTERSTASHSID) into a general-purpose register, use the MRC (or MRC2) instruction with the following syntax:

```
MRC p15, 0, <Rt>, c15, c4, 2; Read CLUSTERSTASHSID into Rt
```

To write this register in AArch32 state (CLUSTERSTASHSID) from a general-purpose register, use the MCR (or MCR2) instruction with the following syntax:

```
MCR p15, 0, <Rt>, c15, c4, 2; Write Rt into CLUSTERSTASHSID
```

Accessibility

This register is accessible in software as follows:

<systemreg>	Control			Accessibility			
	E2H	TGE	NS	EL0	EL1	EL2	EL3
CLUSTERSTASHSID	x	x	0	-	RW	n/a	RW
CLUSTERSTASHSID	x	0	1	-	RW	RW	RW
CLUSTERSTASHSID	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

This register is write accessible in EL1 if ACTLR_EL3.SMEN is 1 and ACTLR_EL2.SMEN is 1, or ACTLR_EL3.SMEN is 1 and SCR.NS is 0.

If write access is not permitted, then Trap to the lowest Exception level that denied access (EL2 or EL3).

11.18 CLUSTERTHREADSID, Cluster Thread Scheme ID Register

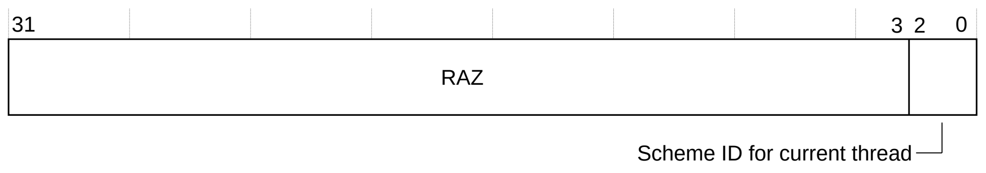
The CLUSTERTHREADSID register is banked for each thread of execution. For multithreaded cores the scheme ID is generated by a logical OR of the thread ID registers for that core.

Bit field descriptions

This description applies to both the AArch32 (CLUSTERTHREADSID) and AArch64 (CLUSTERTHREADSID_EL1) registers.

CLUSTERTHREADSID is a 32-bit register, and is part of SCU and L3 cache configuration registers.

Figure 11-14: CLUSTERTHREADSID bit assignments



RAZ, [31:3]

Read-As-Zero.

Scheme ID for current thread, [2:0]

Scheme ID for this thread.

These bits reset to `0b000`.

Configurations

The AArch32 CLUSTERTHREADSID is architecturally mapped to the AArch64 CLUSTERTHREADSID_EL1 register.

Usage constraints

Accessing the CLUSTERTHREADSID

In AArch64 state (CLUSTERTHREADSID_EL1):

To read this register in AArch64 state (CLUSTERTHREADSID_EL1) into a general-purpose register, use the MRS instruction with the following syntax:

```
MRS <Xt>, S3_0_C15_C4_0; Read CLUSTERTHREADSID_EL1 into Xt
```

To write this register in AArch64 state (CLUSTERTHREADSID_EL1) from a general-purpose register, use the MSR instruction with the following syntax:

```
MSR S3_0_C15_C4_0, <Xt>; Write Xt into CLUSTERTHREADSID_EL1
```

In AArch32 state (CLUSTERTHREADSID):

To read this register in AArch32 state (CLUSTERTHREADSID) into a general-purpose register, use the MRC (or MRC2) instruction with the following syntax:

```
MRC p15, 0, <Rt>, c15, c4, 0; Read CLUSTERTHREADSID into Rt
```

To write this register in AArch32 state (CLUSTERTHREADSID) from a general-purpose register, use the MCR (or MCR2) instruction with the following syntax:

```
MCR p15, 0, <Rt>, c15, c4, 0; Write Rt into CLUSTERTHREADSID
```

Accessibility

This register is accessible in software as follows:

<systemreg>	Control			Accessibility			
	E2H	TGE	NS	EL0	EL1	EL2	EL3
CLUSTERTHREADSID	x	x	0	-	RW	n/a	RW
CLUSTERTHREADSID	x	0	1	-	RW	RW	RW
CLUSTERTHREADSID	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

This register is write accessible in EL1 if ACTLR_EL3.TSIDEN is 1 and ACTLR_EL2.TSIDEN is 1, or ACTLR_EL3.TSIDEN is 1 and SCR.NS is 0.

If write access is not permitted, then Trap to the lowest Exception level that denied access (EL2 or EL3).

11.19 CLUSTERTHREADSIDOVR, Cluster Thread Scheme ID Override Register

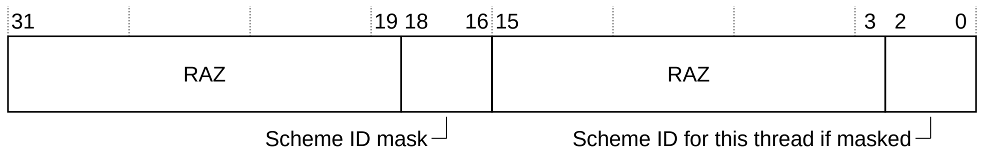
The CLUSTERTHREADSIDOVR register can be used by more privileged software, for example a hypervisor, to partition the scheme IDs. Therefore allowing the less privileged software to use only a subset of scheme IDs. This register is RW. The CLUSTERTHREADSIDOVR register is banked for each thread of execution.

Bit field descriptions

This description applies to both the AArch32 (CLUSTERTHREADSIDOVR) and AArch64 (CLUSTERTHREADSIDOVR_EL1) registers.

CLUSTERTHREADSIDOVR is a 32-bit register, and is part of SCU and L3 cache configuration registers.

Figure 11-15: CLUSTERTHREADSIDOVR bit assignments



RAZ, [31:19]

Read-As-Zero.

Scheme ID mask, [18:16]

A bit set in the mask causes the matching bit to be taken from this register rather than from the CLUSTERTHREADSID register.

These bits reset to `0b000`.

RAZ, [15:3]

Read-As-Zero.

Scheme ID for this thread if masked, [2:0]

Scheme ID for this thread if masked.

These bits reset to `0b000`.

Configurations

The AArch32 CLUSTERTHREADSIDOVR register is architecturally mapped to the AArch64 CLUSTERTHREADSIDOVR_EL1 register.

Usage constraints

Accessing the CLUSTERTHREADSIDOVR

In AArch64 state (CLUSTERTHREADSIDOVR_EL1):

To read this register in AArch64 state (CLUSTERTHREADSIDOVR_EL1) into a general-purpose register, use the MRS instruction with the following syntax:

```
MRS <Xt>, S3_0_C15_C4_7; Read CLUSTERTHREADSIDOVR_EL1 into Xt
```

To write this register in AArch64 state (CLUSTERTHREADSIDOVR_EL1) from a general-purpose register, use the MSR instruction with the following syntax:

```
MSR S3_0_C15_C4_7, <Xt>; Write Xt into CLUSTERTHREADSIDOVR_EL1
```

In AArch32 state (CLUSTERTHREADSIDOVR):

To read this register in AArch32 state (CLUSTERTHREADSIDOVR) into a general-purpose register, use the MRC (or MRC2) instruction with the following syntax:

```
MRC p15, 0, <Rt>, c15, c4, 7; Read CLUSTERTHREADSIDOVR into Rt
```

To write this register in AArch32 state (CLUSTERTHREADSIDOVR) from a general-purpose register, use the MCR (or MCR2) instruction with the following syntax:

```
MCR p15, 0, <Rt>, c15, c4, 7; Write Rt into CLUSTERTHREADSIDOVR
```

Accessibility

This register is accessible in software as follows:

<systemreg>	Control			Accessibility			
	E2H	TGE	NS	EL0	EL1	EL2	EL3
CLUSTERTHREADSIDOVR	x	x	0	-	RW	n/a	RW
CLUSTERTHREADSIDOVR	x	0	1	-	RW	RW	RW
CLUSTERTHREADSIDOVR	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

This register is write accessible in EL1 if ACTLR_EL3.SMEN is 1 and ACTLR_EL2.SMEN is 1, or ACTLR_EL3.SMEN is 1 and SCR.NS is 0.

If write access is not permitted, then Trap to the lowest Exception level that denied access (EL2 or EL3).

12. Error system registers

This chapter describes the ERR1* error registers for the DSU.

12.1 About the error system registers

The error system registers are accessed by both the AArch32 ERX* error registers and the AArch64 ERX* error registers. The ERR1* registers are agnostic to the architectural state. For example, if ERRSELR==1 and ERRSELR_EL1==1, then ERR1PFGF is accessed by both ERXPFGFR and ERXPFGFR_EL1.

The chapter is presented as follows:

Error system register summary

This section identifies the ERR1* error registers and lists the corresponding ERX* registers that can access them.

Register descriptions

The remainder of the chapter provides register descriptions. They are listed in alphabetical order.

Note

The contents of the error system registers are preserved over a Warm reset.

See appropriate core Technical Reference Manual for more information about the ERX* registers.

12.2 Error system register summary

This section lists the DSU ERR1* error record registers that are accessed from the core AArch32 and AArch64 ERX* error registers.

The ERR1* registers are agnostic to the architectural state. For example, if ERRSELR==1 and ERRSELR_EL1==1, then ERR1PFGF is accessed by both ERXPFGFR and ERXPFGFR_EL1.

For a description of the ERX* error registers see the related core documentation.

The following table describes the architectural error record registers in the DSU.

Table 12-1: Architectural error system register summary

Register mnemonic	Size	Register name and description	Access aliases from AArch32 and AArch64
ERR1CTLR	64	12.3 ERR1CTLR, Error Record Control Register on page 134	ERXCTLR, ERXCTLR2, ERXCTLR_EL1
ERR1FR	64	12.4 ERR1FR, Error Record Feature Register on page 136	ERXFR, ERXFR2, ERXFR_EL1
ERR1MISC0	64	12.5 ERR1MISC0, Error Record Miscellaneous Register 0 on page 138	ERXMISC0, ERXMISC1, ERXMISC0_EL1
ERR1MISC1	64	12.6 ERR1MISC1, Error Record Miscellaneous Register 1 on page 140	ERXMISC2 accesses bits [31:0], ERXMISC3 accesses bits [63:32], ERXMISC1_EL1
ERR1STATUS	32	12.10 ERR1STATUS, Error Record Primary Status Register on page 145	ERXSTATUS, ERXSTATUS_EL1

The following table describes the error record registers that are **IMPLEMENTATION DEFINED** in the DSU.

Table 12-2: Implementation defined error system register summary

Register mnemonic	Size	Register name	Access aliases from AArch32 and AArch64
ERR1PFGCDNR	32	12.7 ERR1PFGCDNR, Error Pseudo Fault Generation Count Down Register on page 141	ERXPFGCDNR, ERXPFGCDNR_EL1
ERR1PFGCTLR	32	12.8 ERR1PFGCTLR, Error Pseudo Fault Generation Control Register on page 141	ERXPFGCTLR, ERXPFGCTLR_EL1
ERR1PFGFR	32	12.9 ERR1PFGFR, Error Pseudo Fault Generation Feature Register on page 143	ERR1PFGFR, ERXPFGFR_EL1

12.3 ERR1CTLR, Error Record Control Register

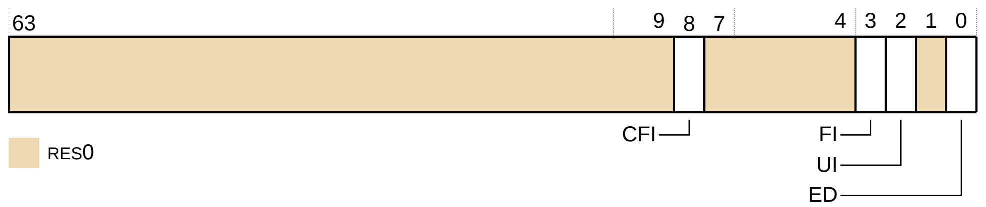
The ERR1CTLR register contains enable bits for the node that writes to this record:

- 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

ERR1CTLR is a 64-bit register and is part of the RAS registers functional group.

Figure 12-1: ERR1CTLR 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 ERR1MISC0 overflows and the overflow bit is set. The possible values are:

0	Fault handling interrupt is not generated for corrected errors.
1	Fault handling interrupt is 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 condition 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 is disabled.
1	Error recovery interrupt enabled.

Note

Applies to both reads and writes.

RES0, [1]

RES0	Reserved.

ED, [0]

Error reporting and logging enable. When reporting is disabled, the DSU will not record or signal errors, but it will continue to detect and correct errors. The possible values of this bit are:

0	Error reporting is disabled. In this state, uncorrectable errors might result in corrupt data being silently propagated.
1	Error reporting is enabled.

Correct error detection and correction codes are written for writes even when reporting is disabled.

Configurations

ERR1CTLR resets to `0x0000000000000000`.

This register is accessible from the following registers when ERRSELR.SEL==1:

- [31:0]: ERXCTLR.
- [63:32]: ERXCTLR2.
- ERXCTLR_EL1.

See the appropriate core Technical Reference Manual for information about these ERX* registers.

12.4 ERR1FR, Error Record Feature Register

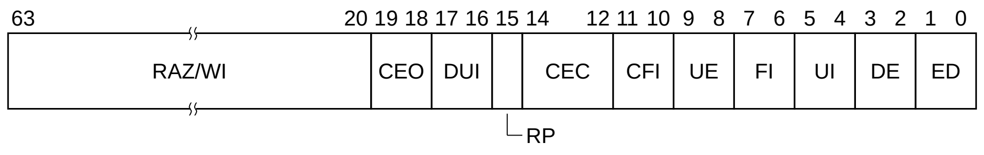
The ERR1FR register defines which of the common architecturally defined features are implemented and, of the implemented features, which are software programmable.

Bit field descriptions

ERR1FR is a 64-bit register, and is part of the *Reliability, Availability, Serviceability* (RAS) registers functional group.

The register is Read Only.

Figure 12-2: ERR1FR bit assignments



RAZ/WI, [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 or cluster 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 are always enabled.

ED, [1:0]

Error detection and correction. The value is:

`0b10`	Error detection is controllable.

Configurations

This register is accessible from the following registers when ERRSELR.SEL==1:

- [31:0]: ERXFR.
- [63:32]: ERXFR2.
- ERXFR_EL1.

See the appropriate core Technical Reference Manual for information about these ERX* registers.

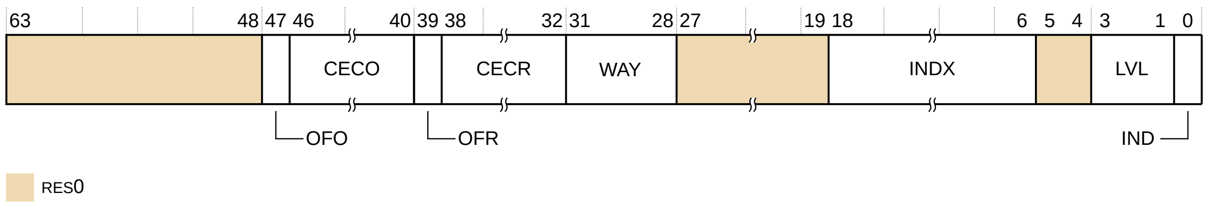
12.5 ERR1MISC0, Error Record Miscellaneous Register 0

The ERR1MISC0 register is an error syndrome register. It contains information on the corrected error count and information to identify where the error was detected. It also contains other state information not present in the corresponding status and address error record registers.

Bit field descriptions

ERR1MISC0 is a 64-bit register, and is part of the *Reliability, Availability, Serviceability* (RAS) registers functional group.

Figure 12-3: ERR1MISC0 bit assignments



RES0, [63:48]

RES0	Reserved.

OFO, [47]

Sticky overflow bit, other. The possible values are:

0	Other counter has not overflowed.
1	Other counter has overflowed.

The fault handling interrupt is generated when the corrected fault handling interrupt is enabled and either overflow bit is set to 1.

CECO, [46:40]

Corrected error count, other. Incremented for each Corrected error that does not match the recorded syndrome.

OFR, [39]

Sticky overflow bit, repeat. The possible values are:

0	Repeat counter has not overflowed.
1	Repeat counter has overflowed.

The fault handling interrupt is generated when the corrected fault handling interrupt is enabled and either overflow bit is set to 1.

CECR, [38:32]

Corrected error count, repeat. Incremented for the first recorded error, which also records other syndrome, and then again for each Corrected error that matches the recorded syndrome.

WAY, [31:28]

Indicates the way in the L3 tag RAM, L3 data RAM, or snoop filter RAM that contained the error. If the error is in the Long-Term Data Buffer (LTDB) RAM, then this indicates the RAM instance that contained the error.

- For the L3 tag RAM, L3 data RAM, and LTDB RAM all four bits are used.
- For the snoop filter RAM, only bits[30:28] are used and bit[31] is set **RES0**.

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 value is:

`0b010`	Level 3.

IND, [0]

Indicates the type of cache that contained the error. The value is:

0	L3 cache.

Configurations

ERR1MISC0 resets to `0x0000000000000000`.

This register is accessible from the following registers when ERRSELR.SEL==1:

- [31:0]: ERXMISC0.
- [63:32]: ERXMISC1.
- ERXMISC0_EL1.

See the appropriate core Technical Reference Manual for information about these ERX* registers.

12.6 ERR1MISC1, Error Record Miscellaneous Register 1

The ERR1MISC1 register is not used in the DSU and marked as **RES0**.

Configurations

ERR1MISC1 is accessible from the following registers when ERRSELR.SEL==1:

- [31:0]: ERXMISC2.
- [63:32]: ERXMISC3.
- ERXMISC1_EL1.

See the appropriate core Technical Reference Manual for information about these ERX* registers.

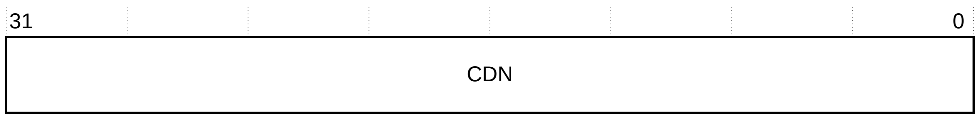
12.7 ERR1PFGCDNR, Error Pseudo Fault Generation Count Down Register

ERR1PFGCDNR is the DSU node register that generates one of the errors enabled in the corresponding ERR1PFGCTL register.

Bit field descriptions

ERR1PFGCDNR is a 32-bit read/write register.

Figure 12-4: ERR1PFGCDNR bit assignments



CDN, [31:0]

Count Down value. The reset value of the Error Generation Counter is used for the countdown.

Configurations

ERR1PFGCDNR resets to `0x00000000`.

There are no configuration options.

ERR1PFGCDNR is accessible from the following registers when ERRSELR.SEL==1:

- ERXPFGCDNR.
- ERXPFGCDNR_EL1.

See the appropriate core Technical Reference Manual for information about these ERX* registers.

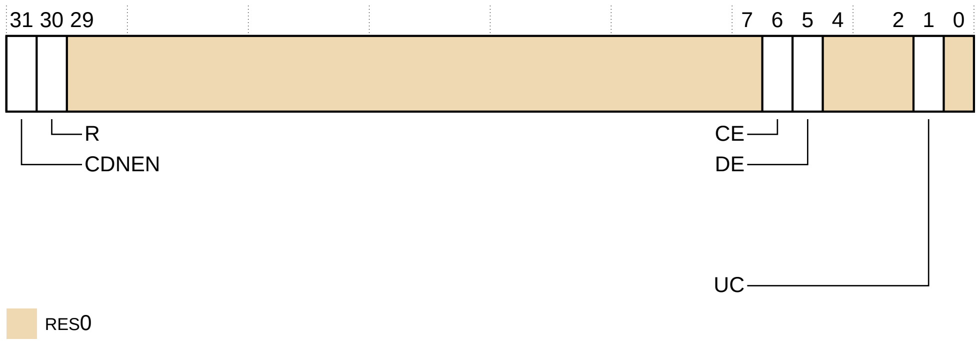
12.8 ERR1PFGCTLR, Error Pseudo Fault Generation Control Register

The ERR1PFGCTLR register is the DSU node register that enables controlled fault generation.

Bit field descriptions

ERR1PFGCTLR is a 32-bit read/write register.

Figure 12-5: ERR1PFGCTLR bit assignments



CDNEN, [31]

Count down enable. This bit controls transfers from the value that is held in the ERR1PFGCDNR 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 ERR1PFGCDNR 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 ERR1PFGCDNR value or stops. The possible values are:

0	When it reaches 0, the counter stops.
1	When it reaches 0, the counter reloads the value that is stored in ERR1PFGCDNR 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 might be generated when the Error Generation Counter is triggered.

DE, [5]

Deferred Error generation enable. The possible values are:

0	No deferred error is generated.
1	A deferred error might be generated when the Error Generation Counter is triggered.

RES0, [4:2]

RES0	Reserved.

UC, [1]

Uncontainable error generation enable. The possible values are:

0	No uncontainable error is generated.
1	An uncontainable error might be generated when the Error Generation Counter is triggered.

RES0, [0]

RES0	Reserved.

Configurations

ERR1PFGCTLR resets to `0x00000000`.

There are no configuration notes.

ERR1PFGCTLR is accessible from the following registers when ERRSELR.SEL==1:

- ERXPFGCTLR.
- ERXPFGCTLR_EL1.

See the appropriate core Technical Reference Manual for information about these ERX* registers.

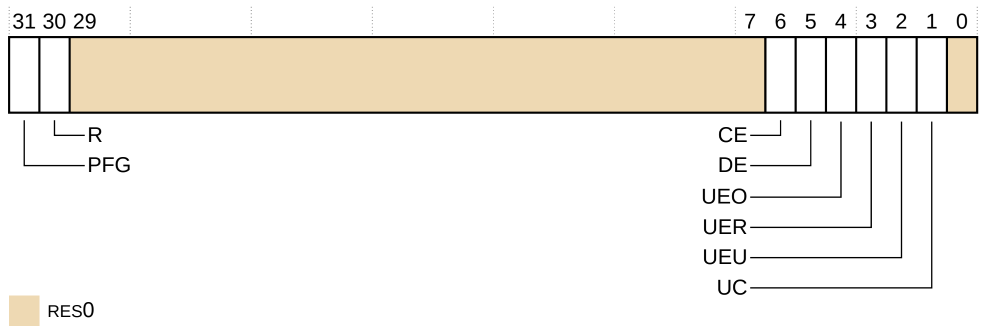
12.9 ERR1PFGFR, Error Pseudo Fault Generation Feature Register

The ERR1PFGFR register is the DSU node register that defines which fault generation features are implemented.

Bit field descriptions

ERR1PFGFR is a 32-bit read-only register.

Figure 12-6: ERR1PFGFR 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.

RES0, [29:7]

RES0	Reserved.

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.
1	This feature is controllable.

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.
1	This feature is controllable.

UC, [1]

Uncontainable Error generation. The possible values are:

0	The node does not support this feature.
1	This feature is controllable.

RES0, [0]

RES0	Reserved.

Configurations

ERR1PFGFR resets to `0xC0000062`.

There are no configuration notes.

ERR1PFGFR is accessible from the following registers when ERRSELR.SEL==1:

- ERXPFGFR.
- ERXPFGFR_EL1.

See the appropriate core Technical Reference Manual for information about these ERX* registers.

12.10 ERR1STATUS, Error Record Primary Status Register

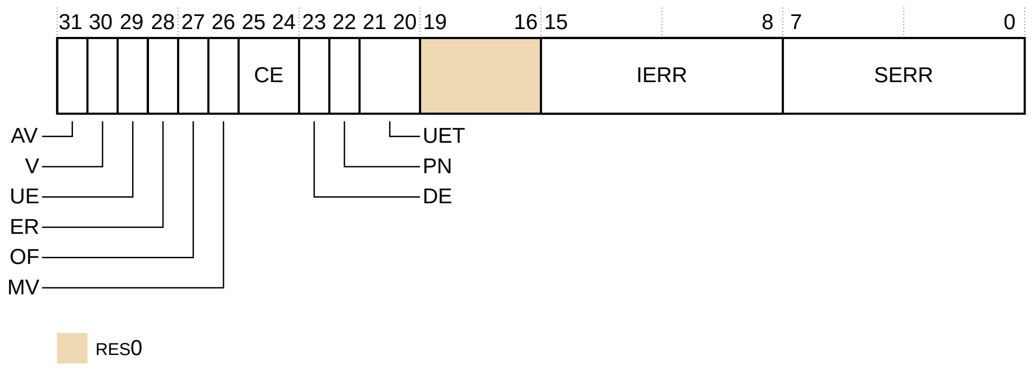
The ERR1STATUS register contains the following information about the error record:

- 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

ERR1STATUS is a 32-bit register.

Figure 12-7: ERR1STATUS bit assignments



AV, [31]

Address Valid. The value is:

0	ERR1ADDR is not valid.

V, [30]

Status Register valid. The possible values are:

0	ERR1STATUS is not valid.
1	ERR1STATUS 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 value is:

0	No external abort has been reported.

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`	ERR1MISC0 and ERR1MISC1 are not valid.
`1`	Indicates that ERR1MISC0 contains additional information about any error recorded that is by this record.

CE, [25:24]

Corrected error. The possible values are:

`0b00`	No corrected errors are 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 possible values are:

`0`	The DSU cannot distinguish a poisoned value from a corrupted value.
`1`	There is an uncorrected error due to data that was earlier poisoned.

UET, [21:20]

Uncorrected Error Type. The value is:

`0b00`	Uncontainable.

RES0, [19:16]

RES0	Reserved.

IERR, [15:8]

An **IMPLEMENTATION DEFINED** error code. The possible values are:

`0x0`	No error, or error on other RAMs.
`0x2`	Error on a L3 snoop filter 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.
`0x12`	Bus error.

Configurations

ERR1STATUS resets to`0x00000000`.

There are no configuration notes.

ERR1STATUS is accessible from the following registers when ERRSELR.SEL==1:

- ERXSTATUS.
- ERXSTATUS_EL1.

See the appropriate core Technical Reference Manual for information about these ERX* registers.

13. PMU registers

This chapter describes the PMU registers for the DSU.

13.1 About the PMU registers

The chapter is presented as follows:

AArch32 PMU register summary
: This section lists the AArch32 PMU registers by access encoding.

AArch64 PMU register summary
: This section lists the AArch64 PMU registers by access encoding.

Register descriptions
: The remainder of the chapter provides generic register descriptions, that apply to both AArch32 and AArch64 registers. They are listed in alphabetical order.

13.2 AArch32 PMU register summary

This section lists the AArch32 PMU registers implemented in the DSU, sorted by access encoding.

Table 13-1: DynamIQ™ Shared Unit AArch32 32-bit PMU registers

Register mnemonic	Copro	CRn	Opc1	CRm	Opc2	Width	Register name and description
CLUSTERPMCR	cp15	c15	0	c5	0	32	See 13.4 CLUSTERPMCR, Cluster Performance Monitors Control Register on page 151
CLUSTERPMCNTENSET	cp15	c15	0	c5	1	32	See 13.5 CLUSTERPMCNTENSET, Cluster Count Enable Set Register on page 154.
CLUSTERPMCNTENCLR	cp15	c15	0	c5	2	32	See 13.6 CLUSTERPMCNTENCLR, Cluster Count Enable Clear Register on page 156.
CLUSTERPMOVSSET	cp15	c15	0	c5	3	32	See 13.7 CLUSTERPMOVSSET, Cluster Overflow Flag Status Set Register on page 158.
CLUSTERPMOVSCLR	cp15	c15	0	c5	4	32	See 13.8 CLUSTERPMOVSCLR, Cluster Overflow Flag Status Clear Register on page 160.
CLUSTERPMSELR	cp15	c15	0	c5	5	32	See 13.9 CLUSTERPMSELR, Cluster Event Counter Selection Register on page 162.
CLUSTERPMINTENSET	cp15	c15	0	c5	6	32	See 13.10 CLUSTERPMINTENSET, Cluster Interrupt Enable Set Register on page 164.
CLUSTERPMINTENCLR	cp15	c15	0	c5	7	32	See 13.11 CLUSTERPMINTENCLR, Cluster Interrupt Enable Clear Register on page 166.
CLUSTERPMCCNTR	cp15	c15	0	c6	0	64-bit register with only bits [31:0] accessible.	See 13.12 CLUSTERPMCCNTR, Cluster Performance Monitors Cycle Counter on page 168.

Register mnemonic	Copro	CRn	Opc1	CRm	Opc2	Width	Register name and description
CLUSTERPMXEVTYPER	cp15	c15	0	c6	1	32	See 13.13 CLUSTERPMXEVTYPER, Cluster Selected Event Type Register on page 169.
CLUSTERPMXEVCNTR	cp15	c15	0	c6	2	32	See 13.14 CLUSTERPMXEVCNTR, Cluster Selected Event Counter Register on page 171.
CLUSTERPMMDCR	cp15	c15	6	c6	3	32	See 13.15 CLUSTERPMMDCR, Cluster Monitor Debug Configuration Register on page 173
CLUSTERPMCEID0	cp15	c15	0	c6	4	32	See 13.16 CLUSTERPMCEID0, Cluster Common Event Identification Register 0 on page 175
CLUSTERPMCEID1	cp15	c15	0	c6	5	32	See 13.17 CLUSTERPMCEID1, Cluster Common Event Identification Register 1 on page 177
CLUSTERPMCLAIMSET	cp15	c15	0	c6	6	32	See 13.18 CLUSTERCLAIMSET, Cluster Claim Tag Set Register on page 179.
CLUSTERPMCLAIMCLR	cp15	c15	0	c6	7	32	See 13.19 CLUSTERCLAIMCLR, Cluster Claim Tag Clear Register on page 181.

The following table describes the DynamIQ™ Shared Unit AArch32 64-bit PMU register.

Table 13-2: DynamIQ™ Shared Unit AArch32 64-bit PMU register

Register mnemonic	Copro	Opc1	CRm	Width	Register name and description
CLUSTERPMCCNTR	cp15	15	c15	64	See 13.12 CLUSTERPMCCNTR, Cluster Performance Monitors Cycle Counter on page 168.

13.3 AArch64 PMU register summary

This section lists the AArch64 PMU registers implemented in the DSU, sorted by access encoding.

Table 13-3: DynamIQ™ Shared Unit AArch64 PMU registers

Register mnemonic	Op0	CRn	Op1	CRm	Op2	Width	Register name and description
CLUSTERPMCR_EL1	3	c15	0	c5	0	32	See 13.4 CLUSTERPMCR, Cluster Performance Monitors Control Register on page 151
CLUSTERPMCNTENSET_EL1	3	c15	0	c5	1	32	See 13.5 CLUSTERPMCNTENSET, Cluster Count Enable Set Register on page 154.
CLUSTERPMCNTENCLR_EL1	3	c15	0	c5	2	32	See 13.6 CLUSTERPMCNTENCLR, Cluster Count Enable Clear Register on page 156.
CLUSTERPMOVSSET_EL1	3	c15	0	c5	3	32	See 13.7 CLUSTERPMOVSSET, Cluster Overflow Flag Status Set Register on page 158.
CLUSTERPMOVSCLR_EL1	3	c15	0	c5	4	32	See 13.8 CLUSTERPMOVSCLR, Cluster Overflow Flag Status Clear Register on page 160.
CLUSTERPMSELR_EL1	3	c15	0	c5	5	32	See 13.9 CLUSTERPMSELR, Cluster Event Counter Selection Register on page 162.
CLUSTERPMINTENSET_EL1	3	c15	0	c5	6	32	See 13.10 CLUSTERPMINTENSET, Cluster Interrupt Enable Set Register on page 164.
CLUSTERPMINTENCLR_EL1	3	c15	0	c5	7	32	See 13.11 CLUSTERPMINTENCLR, Cluster Interrupt Enable Clear Register on page 166.

Register mnemonic	Op0	CRn	Op1	CRm	Op2	Width	Register name and description
CLUSTERPMCCNTR_EL1	3	c15	0	c6	0	64	See 13.12 CLUSTERPMCCNTR, Cluster Performance Monitors Cycle Counter on page 168.
CLUSTERPMXEVTYPER_EL1	3	c15	0	c6	1	32	See 13.13 CLUSTERPMXEVTYPER, Cluster Selected Event Type Register on page 169.
CLUSTERPMXEVCNTR_EL1	3	c15	0	c6	2	32	See 13.14 CLUSTERPMXEVCNTR, Cluster Selected Event Counter Register on page 171.
CLUSTERPMMDCR_EL3	3	c15	6	c6	3	32	See 13.15 CLUSTERPMMDCR, Cluster Monitor Debug Configuration Register on page 173
CLUSTERPMCEID0_EL1	3	c15	0	c6	4	32	See 13.16 CLUSTERPMCEID0, Cluster Common Event Identification Register 0 on page 175.
CLUSTERPMCEID1_EL1	3	c15	0	c6	5	32	See 13.17 CLUSTERPMCEID1, Cluster Common Event Identification Register 1 on page 177.
CLUSTERPMCLAIMSET_EL1	3	c15	0	c6	6	32	See 13.18 CLUSTERCLAIMSET, Cluster Claim Tag Set Register on page 179.
CLUSTERPMCLAIMCLR_EL1	3	c15	0	c6	7	32	See 13.19 CLUSTERCLAIMCLR, Cluster Claim Tag Clear Register on page 181.

13.4 CLUSTERPMCR, Cluster Performance Monitors Control Register

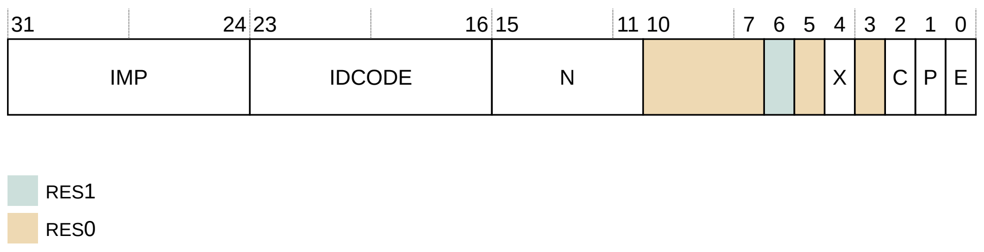
The CLUSTERPMCR register provides details of the Performance Monitors implementation, including the number of counters implemented. CLUSTERPMCR also provides configurations and controls to the counters.

Bit field descriptions

This description applies to both the AArch32 (CLUSTERPMCR) and AArch64 (CLUSTERPMCR_EL1) registers.

CLUSTERPMCR is a 32-bit register, and is part of the PMU registers.

Figure 13-1: CLUSTERPMCR bit assignments



IMP, [31:24]

Indicates the implementer code. This value is:

`0x41`	ASCII character 'A' - implementer is Arm Limited.

This is a read-only field.

IDCODE, [23:16]

Identification code. This value is:

`0x41`	DSU

This is a read-only field.

N, [15:11]

Identifies the number of event counters implemented. The value is:

`0b00110`	Six event counters are implemented together with the cycle counter CLUSTERPMCCNTR.

RES0, [10:7]

RES0	Reserved.

RES1, [6]

RES1	Reserved.

RES0, [5]

RES0	Reserved.

X, [4]

RAZ/WI	Read-As-Zero/Writes ignored.

RES0, [3]

RES0	Reserved.

C, [2]

Clock counter reset. This bit is WO. The effects of writing to this bit are:

0	No action.
1	Reset CLUSTERPMCCNTR to zero.

This bit is always RAZ.

P, [1]

Event counter reset. This bit is WO. The effects of writing to this bit are:

0	No action.
1	Reset all event counters, not including CLUSTERPMCCNTR, to zero.

Resetting the event counters does not clear any overflow bits to 0.

This bit is always RAZ.

E, [0]

Counters enable. The possible values are:

0	All counters including CLUSTERPMCCNTR are disabled.
1	All counters are enabled by CLUSTERPMCNTENSET.

Configurations

The AArch32 CLUSTERPMCR register is architecturally mapped to the AArch64 CLUSTERPMCR_EL1 register.

Usage constraints

Accessing the CLUSTERPMCR

In AArch64 state (CLUSTERPMCR_EL1):

To read this register in AArch64 state (CLUSTERPMCR_EL1) into a general-purpose register, use the MRS instruction with the following syntax:

```
MRS <Xt>, S3_0_C15_C5_0; Read CLUSTERPMCR_EL1 into Xt
```

To write this register in AArch64 state (CLUSTERPMCR_EL1) from a general-purpose register, use the MSR instruction with the following syntax:

```
MSR S3_0_C15_C5_0, <Xt>; Write Xt into CLUSTERPMCR_EL1
```

In AArch32 state (CLUSTERPMCR):

To read this register in AArch32 state (CLUSTERPMCR) into a general-purpose register, use the MRC (or MRC2) instruction with the following syntax:

```
MRC p15, 0, <Rt>, c15, c5, 0; Read CLUSTERPMCR into Rt
```

To write this register in AArch32 state (CLUSTERPMCR) from a general-purpose register, use the MCR (or MCR2) instruction with the following syntax:

```
MCR p15, 0, <Rt>, c15, c5, 0; Write Rt into CLUSTERPMCR
```

Accessibility

This register is accessible in software as follows:

<systemreg>	Control			Accessibility			
	E2H	TGE	NS	EL0	EL1	EL2	EL3
CLUSTERPMCR	x	x	0	-	RW	n/a	RW
CLUSTERPMCR	x	0	1	-	RW	RW	RW
CLUSTERPMCR	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

This register is write accessible in EL1 if ACTLR_EL3[12] is 1 and ACTLR_EL2[12] is 1, or ACTLR_EL3[12] is 1 and SCR.NS is 0. If write access is not possible, then Trap to the lowest Exception level that denied access (EL2 or EL3).

13.5 CLUSTERPMCNTENSET, Cluster Count Enable Set Register

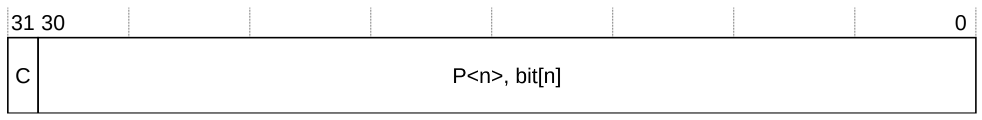
The CLUSTERPMCNTENSET register enables the Cycle Count Register, CLUSTERPMCCNTR, and any implemented event counters CLUSTERPMEVCNTR<n>. Reading this register shows which counters are enabled.

Bit field descriptions

This description applies to both the AArch32 (CLUSTERPMCNTENSET) and AArch64 (CLUSTERPMCNTENSET_EL1) registers.

CLUSTERPMCNTENSET is a 32-bit register, and is part of the PMU registers.

Figure 13-2: CLUSTERPMCNTENSET bit assignments



C, [31]

CLUSTERPMCCNTR enable bit. Enables the cycle counter register. The possible values are:

0	When read, means that the cycle counter is disabled. When written, has no effect.
1	When read, means that the cycle counter is enabled. When written, enables the cycle counter.

P<n>, bit [n], for n = 0-30

Event counter enable bit for CLUSTERPMEVCNTR<n>.

Bits [30:N] are RAZ/WI. N is the value in CLUSTERPMCR.N.

The possible values are:

0	When read, means that CLUSTERPMEVCNTR<n> is disabled. When written, has no effect.

1	When read, means that CLUSTERPMEVCNTR<n> event counter is enabled. When written, enables CLUSTERPMEVCNTR<n>.

Configurations

The AArch32 CLUSTERPMCNTENSET register is architecturally mapped to the AArch64 CLUSTERPMCNTENSET_EL1 register.

Usage constraints

Accessing the CLUSTERPMCNTENSET

In AArch64 state (CLUSTERPMCNTENSET_EL1):

To read this register in AArch64 state (CLUSTERPMCNTENSET_EL1) into a general-purpose register, use the MRS instruction with the following syntax:

```
MRS <Xt>, S3_0_C15_C5_1; Read CLUSTERPMCNTENSET_EL1 into Xt
```

To write this register in AArch64 state (CLUSTERPMCNTENSET_EL1) from a general-purpose register, use the MSR instruction with the following syntax:

```
MSR S3_0_C15_C5_1, <Xt>; Write Xt into CLUSTERPMCNTENSET_EL1
```

In AArch32 state (CLUSTERPMCNTENSET):

To read this register in AArch32 state (CLUSTERPMCNTENSET) into a general-purpose register, use the MRC (or MRC2) instruction with the following syntax:

```
MRC p15, 0, <Rt>, c15, c5, 1; Read CLUSTERPMCNTENSET into Rt
```

To write this register in AArch32 state (CLUSTERPMCNTENSET) from a general-purpose register, use the MCR (or MCR2) instruction with the following syntax:

```
MCR p15, 0, <Rt>, c15, c5, 1; Write Rt into CLUSTERPMCNTENSET
```

Accessibility

This register is accessible in software as follows:

<systemreg>	Control			Accessibility			
	E2H	TGE	NS	EL0	EL1	EL2	EL3
CLUSTERPMCNTENSET	x	x	0	-	RW	n/a	RW
CLUSTERPMCNTENSET	x	0	1	-	RW	RW	RW
CLUSTERPMCNTENSET	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

This register is write accessible in EL1 if ACTLR_EL3[12] is 1 and ACTLR_EL2[12] is 1, or ACTLR_EL3[12] is 1 and SCR.NS is 0. If write access is not possible, then Trap to the lowest Exception level that denied access (EL2 or EL3).

13.6 CLUSTERPMCNTENCLR, Cluster Count Enable Clear Register

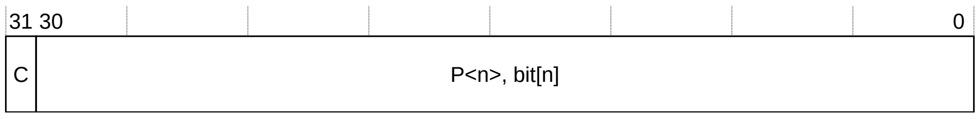
The CLUSTERPMCNTENCLR register disables the Cycle Count Register, CLUSTERPMCCNTR, and any implemented event counters CLUSTERPMEVCNTR<n>. Reading this register shows which counters are enabled. CLUSTERPMCNTENCLR is used along with CLUSTERPMCNTENSET register.

Bit field descriptions

This description applies to both the AArch32 (CLUSTERPMCNTENCLR) and AArch64 (CLUSTERPMCNTENCLR_EL1) registers.

CLUSTERPMCNTENCLR is a 32-bit register, and is part of the PMU registers.

Figure 13-3: CLUSTERPMCNTENCLR bit assignments



C, [31]

CLUSTERPMCCNTR enable bit. Enables the cycle counter register. The possible values are:

0	When read, means that the cycle counter is disabled. When written, has no effect.
1	When read, means that the cycle counter is enabled. When written, disables the cycle counter.

P<n>, bit [n], for n = 0-30

Event counter disable bit for CLUSTERPMEVCNTR<n>.

Bits [30:N] are RAZ/WI. N is the value in CLUSTERPMCR.N.

The possible values are:

0	When read, means that CLUSTERPMEVCNTR<n> is disabled. When written, has no effect.
1	When read, means that CLUSTERPMEVCNTR<n> event counter is enabled. When written, disables CLUSTERPMEVCNTR<n>.

Configurations

The AArch32 CLUSTERPMCNTENCLR register is architecturally mapped to the AArch64 CLUSTERPMCNTENCLR_EL1 register.

Usage constraints

Accessing the CLUSTERPMCNTENCLR

In AArch64 state (CLUSTERPMCNTENCLR_EL1):

To read this register in AArch64 state (CLUSTERPMCNTENCLR_EL1) into a general-purpose register, use the MRS instruction with the following syntax:

```
MRS <Xt>, S3_0_C15_C5_2; Read CLUSTERPMCNTENCLR_EL1 into Xt
```

To write this register in AArch64 state (CLUSTERPMCNTENCLR_EL1) from a general-purpose register, use the MSR instruction with the following syntax:

```
MSR S3_0_C15_C5_2, <Xt>; Write Xt into CLUSTERPMCNTENCLR_EL1
```

In AArch32 state (CLUSTERPMCNTENCLR):

To read this register in AArch32 state (CLUSTERPMCNTENCLR) into a general-purpose register, use the MRC (or MRC2) instruction with the following syntax:

```
MRC p15, 0, <Rt>, c15, c5, 2; Read CLUSTERPMCNTENCLR into Rt
```

To write this register in AArch32 state (CLUSTERPMCNTENCLR) from a general-purpose register, use the MCR (or MCR2) instruction with the following syntax:

```
MCR p15, 0, <Rt>, c15, c5, 2; Write Rt into CLUSTERPMCNTENCLR
```

Accessibility

This register is accessible in software as follows:

<systemreg>	Control			Accessibility			
	E2H	TGE	NS	EL0	EL1	EL2	EL3
CLUSTERPMCNTENCLR	x	x	0	-	RW	n/a	RW
CLUSTERPMCNTENCLR	x	0	1	-	RW	RW	RW
CLUSTERPMCNTENCLR	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

This register is write accessible in EL1 if ACTLR_EL3[12] is 1 and ACTLR_EL2[12] is 1, or ACTLR_EL3[12] is 1 and SCR.NS is 0. If write access is not possible, then Trap to the lowest Exception level that denied access (EL2 or EL3).

13.7 CLUSTERPMOVSSET, Cluster Overflow Flag Status Set Register

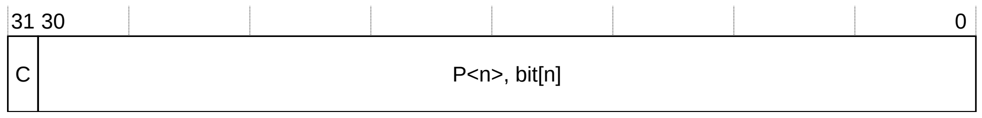
The CLUSTERPMOVSSET register sets the state of the overflow bit for the Cycle Count Register, CLUSTERPMCCNTR, and each of the implemented event counters CLUSTERPMEVCNTR<n>.

Bit field descriptions

This description applies to both the AArch32 (CLUSTERPMOVSSET) and AArch64 (CLUSTERPMOVSSET_EL1) registers.

CLUSTERPMOVSSET is a 32-bit register, and is part of the PMU registers.

Figure 13-4: CLUSTERPMOVSSET bit assignments



C, [31]

CLUSTERPMCCNTR overflow bit. The possible values are:

0	When read, means that the cycle counter has not overflowed. When written, has no effect.
1	When read, means that the cycle counter has overflowed. When written, sets the overflow bit to 1.

P<n>, bit [n], for n = 0-30

Event counter overflow set bit for CLUSTERPMEVCNTR<n>.

Bits [30:N] are RAZ/WI. N is the value in CLUSTERPMCR.N.

The possible values are:

0	When read, means that CLUSTERPMEVCNTR<n> has not overflowed. When written, has no effect.
1	When read, means that CLUSTERPMEVCNTR<n> has overflowed. When written, sets the CLUSTERPMEVCNTR<n> overflow bit to 1.

Configurations

The AArch32 CLUSTERPMOVSSET register is architecturally mapped to the AArch64 CLUSTERPMOVSSET_EL1 register.

Usage constraints

Accessing the CLUSTERPMOVSSET

In AArch64 state (CLUSTERPMOVSSET_EL1):

To read this register in AArch64 state (CLUSTERPMOVSSET_EL1) into a general-purpose register, use the MRS instruction with the following syntax:

```
MRS <Xt>, S3_0_C15_C5_3; Read CLUSTERPMOVSSET_EL1 into Xt
```

To write this register in AArch64 state (CLUSTERPMOVSSET_EL1) from a general-purpose register, use the MSR instruction with the following syntax:

```
MSR S3_0_C15_C5_3, <Xt>; Write Xt into CLUSTERPMOVSSET_EL1
```

In AArch32 state (CLUSTERPMOVSSET):

To read this register in AArch32 state (CLUSTERPMOVSSET) into a general-purpose register, use the MRC (or MRC2) instruction with the following syntax:

```
MRC p15, 0, <Rt>, c15, c5, 3; Read CLUSTERPMOVSSET into Rt
```

To write this register in AArch32 state (CLUSTERPMOVSSET) from a general-purpose register, use the MCR (or MCR2) instruction with the following syntax:

```
MCR p15, 0, <Rt>, c15, c5, 3; Write Rt into CLUSTERPMOVSSET
```

Accessibility

This register is accessible in software as follows:

<systemreg>	Control			Accessibility			
	E2H	TGE	NS	EL0	EL1	EL2	EL3
CLUSTERPMOVSSET	x	x	0	-	RW	n/a	RW
CLUSTERPMOVSSET	x	0	1	-	RW	RW	RW
CLUSTERPMOVSSET	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

This register is write accessible in EL1 if ACTLR_EL3[12] is 1 and ACTLR_EL2[12] is 1, or ACTLR_EL3[12] is 1 and SCR.NS is 0. If write access is not possible, then Trap to the lowest Exception level that denied access (EL2 or EL3).

13.8 CLUSTERPMOVSCLR, Cluster Overflow Flag Status Clear Register

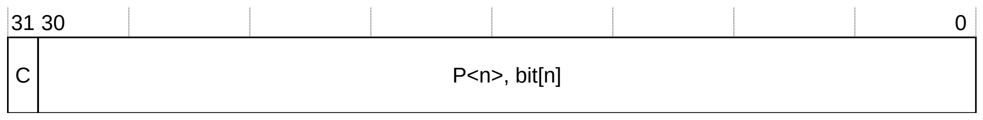
The CLUSTERPMOVSCLR register contains the state of the overflow bit for the Cycle Counter Register, CLUSTERPMCCNTR, and each of the implemented event counters CLUSTERPMEVCNTR<n>. Writing to this register clears these bits.

Bit field descriptions

This description applies to both the AArch32 (CLUSTERPMOVSCLR) and AArch64 (CLUSTERPMOVSCLR_EL1) registers.

CLUSTERPMOVSCLR is a 32-bit register, and is part of the PMU registers.

Figure 13-5: CLUSTERPMOVSCLR bit assignments



C, [31]

CLUSTERPMCCNTR overflow bit. The possible values are:

0	When read, means that the cycle counter has not overflowed. When written, has no effect.
1	When read, means that the cycle counter has overflowed. When written, clears the overflow bit to 0.

CLUSTERPMCR.LC controls whether an overflow is detected from CLUSTERPMCCNTR [31] or from CLUSTERPMCCNTR.

P<n>, bit [n], for n = 0-30

Event counter overflow clear bit for CLUSTERPMEVCNTR<n>.

Bits [30:N] are RAZ/WI. N is the value in CLUSTERPMCR.N.

The possible values are:

0	When read, means that CLUSTERPMEVCNTR<n> has not overflowed. When written, has no effect.
1	When read, means that CLUSTERPMEVCNTR<n> has overflowed. When written, clears the CLUSTERPMEVCNTR<n> overflow bit to 0.

Configurations

The AArch32 CLUSTERPMOVSCLR register is architecturally mapped to the AArch64 CLUSTERPMOVSCLR_EL1 register.

Usage constraints

Accessing the CLUSTERPMOVSCLR

In AArch64 state (CLUSTERPMOVSCLR_EL1):

To read this register in AArch64 state (CLUSTERPMOVSCLR_EL1) into a general-purpose register, use the MRS instruction with the following syntax:

```
MRS <Xt>, S3_0_C15_C5_4; Read CLUSTERPMOVSCLR_EL1 into Xt
```

To write this register in AArch64 state (CLUSTERPMOVSCLR_EL1) from a general-purpose register, use the MSR instruction with the following syntax:

```
MSR S3_0_C15_C5_4, <Xt>; Write Xt into CLUSTERPMOVSCLR_EL1
```

In AArch32 state (CLUSTERPMOVSCLR):

To read this register in AArch32 state (CLUSTERPMOVSCLR) into a general-purpose register, use the MRC (or MRC2) instruction with the following syntax:

```
MRC p15, 0, <Rt>, c15, c5, 4; Read CLUSTERPMOVSCLR into Rt
```

To write this register in AArch32 state (CLUSTERPMOVSCLR) from a general-purpose register, use the MCR (or MCR2) instruction with the following syntax:

```
MCR p15, 0, <Rt>, c15, c5, 4; Write Rt into CLUSTERPMOVSCLR
```

Accessibility

This register is accessible in software as follows:

<systemreg>	Control			Accessibility			
	E2H	TGE	NS	EL0	EL1	EL2	EL3
CLUSTERPMOVSCLR	x	x	0	-	RW	n/a	RW
CLUSTERPMOVSCLR	x	0	1	-	RW	RW	RW
CLUSTERPMOVSCLR	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

This register is write accessible in EL1 if ACTLR_EL3[12] is 1 and ACTLR_EL2[12] is 1, or ACTLR_EL3[12] is 1 and SCR.NS is 0. If write access is not possible, then Trap to the lowest Exception level that denied access (EL2 or EL3).

13.9 CLUSTERPMSELR, Cluster Event Counter Selection Register

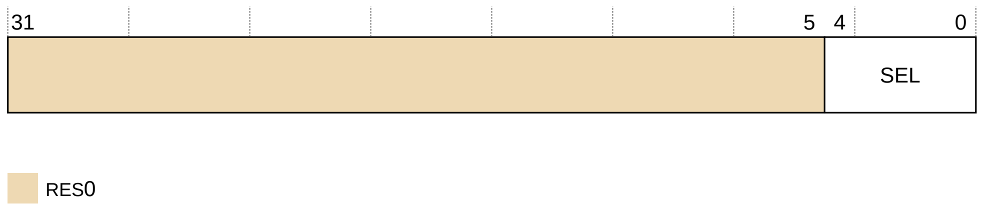
The CLUSTERPMSELR register selects the current event counter CLUSTERPMEVCNTR<n>. CLUSTERPMSELR is used along with CLUSTERPMXEVTYPER to determine the value of a selected event counter. It is also used along with CLUSTERPMXEVCNTR, to determine the value of a selected event counter.

Bit field descriptions

This description applies to both the AArch32 (CLUSTERPMSELR) and AArch64 (CLUSTERPMSELR_EL1) registers.

CLUSTERPMSELR is a 32-bit register, and is part of the PMU registers.

Figure 13-6: CLUSTERPMSELR bit assignments



RES0, [31:5]

RES0	Reserved.

SEL, [4:0]

Selects event counter, CLUSTERPMEVCNTR<n>, where n is the value held in this field. This value identifies which event counter is accessed when a subsequent access to CLUSTERPMXEVTYPER or CLUSTERPMXEVCNTR occurs.

This field can take any value from `0b00000` to (CLUSTERPMCR.N - 1).

If this field is set to a value greater than or equal to the number of implemented counters, the results of access to CLUSTERPMXEVTYPER or CLUSTERPMXEVCNTR are as if the register is RAZ/WI.

Direct reads of this field return an UNKNOWN value.

Configurations

The AArch32 CLUSTERPMSELR register is architecturally mapped to the AArch64 CLUSTERPMSELR_EL1 register.

Usage constraints

Accessing the CLUSTERPMSELR

In AArch64 state (CLUSTERPMSELR_EL1):

To read this register in AArch64 state (CLUSTERPMSELR_EL1) into a general-purpose register, use the MRS instruction with the following syntax:

```
MRS <Xt>, S3_0_C15_C5_5; Read CLUSTERPMSELR_EL1 into Xt
```

To write this register in AArch64 state (CLUSTERPMSELR_EL1) from a general-purpose register, use the MSR instruction with the following syntax:

```
MSR S3_0_C15_C5_5, <Xt>; Write Xt into CLUSTERPMSELR_EL1
```

In AArch32 state (CLUSTERPMSELR):

To read this register in AArch32 state (CLUSTERPMSELR) into a general-purpose register, use the MRC (or MRC2) instruction with the following syntax:

```
MRC p15, 0, <Rt>, c15, c5, 5; Read CLUSTERPMSELR into Rt
```

To write this register in AArch32 state (CLUSTERPMSELR) from a general-purpose register, use the MCR (or MCR2) instruction with the following syntax:

```
MCR p15, 0, <Rt>, c15, c5, 5; Write Rt into CLUSTERPMSELR
```

Accessibility

This register is accessible in software as follows:

<systemreg>	Control			Accessibility			
	E2H	TGE	NS	EL0	EL1	EL2	EL3
CLUSTERPMSELR	x	x	0	-	RW	n/a	RW
CLUSTERPMSELR	x	0	1	-	RW	RW	RW
CLUSTERPMSELR	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

This register is write accessible in EL1 if ACTLR_EL3[12] is 1 and ACTLR_EL2[12] is 1, or ACTLR_EL3[12] is 1 and SCR.NS is 0. If write access is not possible, then Trap to the lowest Exception level that denied access (EL2 or EL3).

13.10 CLUSTERPMINTENSET, Cluster Interrupt Enable Set Register

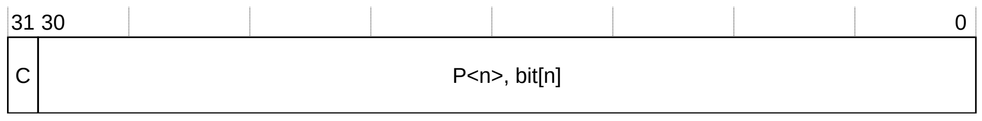
The CLUSTERPMINTENSET register enables the generation of interrupt requests on overflows from the Cycle Count Register, CLUSTERPMCCNTR, and the event counters, CLUSTERPMEVCNTR<n>. Reading the register shows which overflow interrupt requests are enabled.

Bit field descriptions

This description applies to both the AArch32 (CLUSTERPMINTENSET) and AArch64 (CLUSTERPMINTENSET_EL1) registers.

CLUSTERPMINTENSET is a 32-bit register, and is part of the PMU registers.

Figure 13-7: CLUSTERPMINTENSET bit assignments



C, [31]

CLUSTERPMCCNTR overflow interrupt request enable bit. The possible values are:

0	When read, means that the cycle counter overflow interrupt request is disabled. When written, has no effect.
1	When read, means that the cycle counter overflow interrupt request is enabled. When written, enables the cycle count overflow interrupt request.

P<n>, bit [n], for n = 0-30

Event counter overflow interrupt request enable bit for CLUSTERPMEVCNTR<n>.

Bits [30:N] are RAZ/WI. N is the value in CLUSTERPMCR.N.

The possible values are:

0	When read, means that CLUSTERPMEVCNTR<n> event counter interrupt request is disabled. When written, has no effect.

1	When read, means that CLUSTERPMEVCNTR<n> event counter interrupt request is enabled. When written, enables the CLUSTERPMEVCNTR<n> interrupt request.

Configurations

The AArch32 CLUSTERPMINTENSET register is architecturally mapped to the AArch64 CLUSTERPMINTENSET_EL1 register.

Usage constraints

Accessing the CLUSTERPMINTENSET

In AArch64 state (CLUSTERPMINTENSET_EL1):

To read this register in AArch64 state (CLUSTERPMINTENSET_EL1) into a general-purpose register, use the MRS instruction with the following syntax:

```
MRS <Xt>, S3_0_C15_C5_6; Read CLUSTERPMINTENSET_EL1 into Xt
```

To write this register in AArch64 state (CLUSTERPMINTENSET_EL1) from a general-purpose register, use the MSR instruction with the following syntax:

```
MSR S3_0_C15_C5_6, <Xt>; Write Xt into CLUSTERPMINTENSET_EL1
```

In AArch32 state (CLUSTERPMINTENSET):

To read this register in AArch32 state (CLUSTERPMINTENSET) into a general-purpose register, use the MRC (or MRC2) instruction with the following syntax:

```
MRC p15, 0, <Rt>, c15, c5, 6; Read CLUSTERPMINTENSET into Rt
```

To write this register in AArch32 state (CLUSTERPMINTENSET) from a general-purpose register, use the MCR (or MCR2) instruction with the following syntax:

```
MCR p15, 0, <Rt>, c15, c5, 6; Write Rt into CLUSTERPMINTENSET
```

Accessibility

This register is accessible in software as follows:

<systemreg>	Control			Accessibility			
	E2H	TGE	NS	EL0	EL1	EL2	EL3
CLUSTERPMINTENSET	x	x	0	-	RW	n/a	RW
CLUSTERPMINTENSET	x	0	1	-	RW	RW	RW
CLUSTERPMINTENSET	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

This register is write accessible in EL1 if ACTLR_EL3[12] is 1 and ACTLR_EL2[12] is 1, or ACTLR_EL3[12] is 1 and SCR.NS is 0. If write access is not possible, then Trap to the lowest Exception level that denied access (EL2 or EL3).

13.11 CLUSTERPMINTENCLR, Cluster Interrupt Enable Clear Register

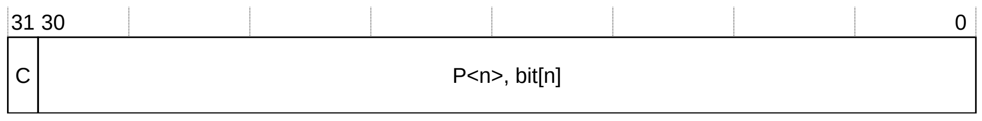
The CLUSTERPMINTENCLR register disables the generation of interrupt requests on overflows from the Cycle Count Register, CLUSTERPMCCNTR, and the event counters, CLUSTERPMEVCNTR<n>. Reading the register shows which overflow interrupt requests are enabled.

Bit field descriptions

This description applies to both the AArch32 (CLUSTERPMINTENCLR) and AArch64 (CLUSTERPMINTENCLR_EL1) registers.

CLUSTERPMINTENCLR is a 32-bit register, and is part of the PMU registers.

Figure 13-8: CLUSTERPMINTENCLR bit assignments



C, [31]

CLUSTERPMCCNTR overflow interrupt request disable bit. The possible values are:

0	When read, means that the cycle counter overflow interrupt request is disabled. When written, has no effect.
1	When read, means that the cycle counter overflow interrupt request is enabled. When written, disables the cycle count overflow interrupt request.

P<n>, bit [n], for n = 0-30

Event counter overflow interrupt request disable bit for CLUSTERPMEVCNTR<n>.

Bits [30:N] are RAZ/WI. N is the value in CLUSTERPMCR.N.

The possible values are:

0	When read, means that CLUSTERPMEVCNTR<n> event counter interrupt request is disabled. When written, has no effect.

1	When read, means that CLUSTERPMEVCNTR<n> event counter interrupt request is enabled. When written, disables the CLUSTERPMEVCNTR<n> interrupt request.

Configurations

The AArch32 CLUSTERPMINTENCLR register is architecturally mapped to the AArch64 CLUSTERPMINTENCLR_EL1 register.

Usage constraints

Accessing the CLUSTERPMINTENCLR

In AArch64 state (CLUSTERPMINTENCLR_EL1):

To read this register in AArch64 state (CLUSTERPMINTENCLR_EL1) into a general-purpose register, use the MRS instruction with the following syntax:

```
MRS <Xt>, S3_0_C15_C5_7; Read CLUSTERPMINTENCLR_EL1 into Xt
```

To write this register in AArch64 state (CLUSTERPMINTENCLR_EL1) from a general-purpose register, use the MSR instruction with the following syntax:

```
MSR S3_0_C15_C5_7, <Xt>; Write Xt into CLUSTERPMINTENCLR_EL1
```

In AArch32 state (CLUSTERPMINTENCLR):

To read this register in AArch32 state (CLUSTERPMINTENCLR) into a general-purpose register, use the MRC (or MRC2) instruction with the following syntax:

```
MRC p15, 0, <Rt>, c15, c5, 7; Read CLUSTERPMINTENCLR into Rt
```

To write this register in AArch32 state (CLUSTERPMINTENCLR) from a general-purpose register, use the MCR (or MCR2) instruction with the following syntax:

```
MCR p15, 0, <Rt>, c15, c5, 7; Write Rt into CLUSTERPMINTENCLR
```

Accessibility

This register is accessible in software as follows:

<systemreg>	Control			Accessibility			
	E2H	TGE	NS	EL0	EL1	EL2	EL3
CLUSTERPMINTENCLR	x	x	0	-	RW	n/a	RW
CLUSTERPMINTENCLR	x	0	1	-	RW	RW	RW
CLUSTERPMINTENCLR	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

This register is write accessible in EL1 if ACTLR_EL3[12] is 1 and ACTLR_EL2[12] is 1, or ACTLR_EL3[12] is 1 and SCR.NS is 0. If write access is not possible, then Trap to the lowest Exception level that denied access (EL2 or EL3).

13.12 CLUSTERPMCCNTR, Cluster Performance Monitors Cycle Counter

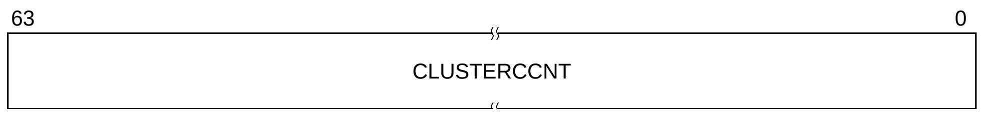
The CLUSTERPMCCNTR register holds the value of the Cluster Cycle Counter, CLUSTERCCNT, that counts cluster clock cycles.

Bit field descriptions

This description applies to both the AArch32 (CLUSTERPMCCNTR) and AArch64 (CLUSTERPMCCNTR_EL1) registers.

CLUSTERPMCCNTR is a 64-bit register, and is part of the PMU registers.

Figure 13-9: CLUSTERPMCCNTR bit assignments



CLUSTERCCNT, [63:0]

Cluster cycle count. This field increments every cluster clock cycle.

Writing 1 to CLUSTERPMCR.C sets this field to 0.

Configurations

The AArch32 CLUSTERPMCCNTR register is architecturally mapped to the AArch64 CLUSTERPMCCNTR_EL1[63:0] register.

All counters are subject to any changes in clock frequency, including clock stopping caused by entering quiescent states. This means that it is CONSTRAINED UNPREDICTABLE whether or not CLUSTERPMCCNTR continues to increment when clocks are stopped.

This register is in the Warm reset domain. On a Warm or Cold reset RW fields in this register reset to architecturally UNKNOWN values.

Usage constraints

Accessing the CLUSTERPMCCNTR

In AArch64 state (CLUSTERPMCCNTR_EL1):

To read this register in AArch64 state (CLUSTERPMCCNTR_EL1) into a general-purpose register, use the MRS instruction with the following syntax:

```
MRS <Xt>, S3_0_C15_C6_0; Read CLUSTERPMCCNTR_EL1 into Xt
```

To write this register in AArch64 state (CLUSTERPMCCNTR_EL1) from a general-purpose register, use the MSR instruction with the following syntax:

```
MSR S3_0_C15_C6_0, <Xt>; Write Xt into CLUSTERPMCCNTR_EL1
```

In AArch32 state (CLUSTERPMCCNTR):

To read this register in AArch32 state (CLUSTERPMCCNTR) into a general-purpose register, use the MRC (or MRC2) instruction with the following syntax:

```
MRC p15, 0, <Rt>, c15, c6, 0; Read CLUSTERPMCCNTR into Rt
```

To write this register in AArch32 state (CLUSTERPMCCNTR) from a general-purpose register, use the MCR (or MCR2) instruction with the following syntax:

```
MCR p15, 0, <Rt>, c15, c6, 0; Write Rt into CLUSTERPMCCNTR
```

Accessibility

This register is accessible in software as follows:

<systemreg>	Control			Accessibility			
	E2H	TGE	NS	EL0	EL1	EL2	EL3
CLUSTERPMCCNTR	x	x	0	-	RW	n/a	RW
CLUSTERPMCCNTR	x	0	1	-	RW	RW	RW
CLUSTERPMCCNTR	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

This register is write accessible in EL1 if ACTLR_EL3[12] is 1 and ACTLR_EL2[12] is 1, or ACTLR_EL3[12] is 1 and SCR.NS is 0. If write access is not possible, then Trap to the lowest Exception level that denied access (EL2 or EL3).

13.13 CLUSTERPMXEVTYPER, Cluster Selected Event Type Register

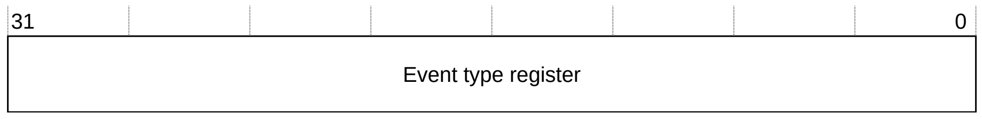
When CLUSTERPMSELR.SEL selects an event counter, CLUSTERPMXEVTYPER accesses a CLUSTERPMEVTYPER<n> register.

Bit field descriptions

This description applies to both the AArch32 (CLUSTERPMXEVTYPER) and AArch64 (CLUSTERPMXEVTYPER_EL1) registers.

CLUSTERPMXEVTYPER is a 32-bit register, and is part of the PMU registers.

Figure 13-10: CLUSTERPMXEVTYPER bit assignments



Bits, [31:0]

Event type register.

This register accesses CLUSTERPMEVTYPER<n> where n is the value in CLUSTERPMSELR.SEL.

Configurations

The AArch32 CLUSTERPMXEVTYPER register is architecturally mapped to the AArch64 CLUSTERPMXEVTYPER_EL1 register.

This register is in the Warm reset domain. On a Warm or Cold reset RW fields in this register reset to architecturally UNKNOWN values.

Usage constraints

Accessing the CLUSTERPMXEVTYPER

In AArch64 state (CLUSTERPMXEVTYPER_EL1):

To read this register in AArch64 state (CLUSTERPMXEVTYPER_EL1) into a general-purpose register, use the MRS instruction with the following syntax:

```
MRS <Xt>, S3_0_C15_C6_1; Read CLUSTERPMXEVTYPER_EL1 into Xt
```

To write this register in AArch64 state (CLUSTERPMXEVTYPER_EL1) from a general-purpose register, use the MSR instruction with the following syntax:

```
MSR S3_0_C15_C6_1, <Xt>; Write Xt into CLUSTERPMXEVTYPER_EL1
```

In AArch32 state (CLUSTERPMXEVTYPER):

To read this register in AArch32 state (CLUSTERPMXEVTYPER) into a general-purpose register, use the MRC (or MRC2) instruction with the following syntax:

```
MRC p15, 0, <Rt>, c15, c6, 1; Read CLUSTERPMXEVTYPER into Rt
```

To write this register in AArch32 state (CLUSTERPMXEVTYPER) from a general-purpose register, use the MCR (or MCR2) instruction with the following syntax:

```
MCR p15, 0, <Rt>, c15, c6, 1; Write Rt into CLUSTERPMXEVTYPER
```

Accessibility

This register is accessible in software as follows:

<systemreg>	Control			Accessibility			
	E2H	TGE	NS	EL0	EL1	EL2	EL3
CLUSTERPMXEVTYPER	x	x	0	-	RW	n/a	RW
CLUSTERPMXEVTYPER	x	0	1	-	RW	RW	RW
CLUSTERPMXEVTYPER	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

This register is write accessible in EL1 if ACTLR_EL3[12] is 1 and ACTLR_EL2[12] is 1, or ACTLR_EL3[12] is 1 and SCR.NS is 0. If write access is not possible, then Trap to the lowest Exception level that denied access (EL2 or EL3).

Related information

CLUSTERPMEVTYPER<n>, Cluster Event Type Register on page 183

13.14 CLUSTERPMXEVCNTR, Cluster Selected Event Counter Register

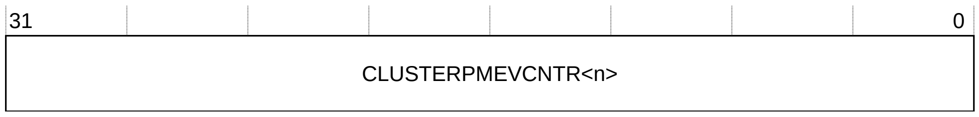
The CLUSTERPMXEVCNTR register reads or writes the values of the selected event counter, CLUSTERPMEVCNTR<n>. The register field CLUSTERPMSELR.SEL determines which event counter is selected.

Bit field descriptions

This description applies to both the AArch32 (CLUSTERPMXEVCNTR) and AArch64 (CLUSTERPMXEVCNTR_EL1) registers.

CLUSTERPMXEVCNTR is a 32-bit register, and is part of the PMU registers.

Figure 13-11: CLUSTERPMXEVCNTR bit assignments



CLUSTERPMEVCNTR<n>, bits [31:0]

Value of the selected event counter, CLUSTERPMEVCNTR<n>, where n is the value stored in CLUSTERPMSELR.SEL.

Configurations

The AArch32 CLUSTERPMXEVCNTR register is architecturally mapped to the AArch64 CLUSTERPMXEVCNTR_EL1 register.

This register is in the Warm reset domain. On a Warm or Cold reset RW fields in this register reset to architecturally UNKNOWN values.

Usage constraints

Accessing the CLUSTERPMXEVCNTR

In AArch64 state (CLUSTERPMXEVCNTR_EL1):

To read this register in AArch64 state (CLUSTERPMXEVCNTR_EL1) into a general-purpose register, use the MRS instruction with the following syntax:

```
MRS <Xt>, S3_0_C15_C6_2; Read CLUSTERPMXEVCNTR_EL1 into Xt
```

To write this register in AArch64 state (CLUSTERPMXEVCNTR_EL1) from a general-purpose register, use the MSR instruction with the following syntax:

```
MSR S3_0_C15_C6_2, <Xt>; Write Xt into CLUSTERPMXEVCNTR_EL1
```

In AArch32 state (CLUSTERPMXEVCNTR):

To read this register in AArch32 state (CLUSTERPMXEVCNTR) into a general-purpose register, use the MRC (or MRC2) instruction with the following syntax:

```
MRC p15, 0, <Rt>, c15, c6, 2; Read CLUSTERPMXEVCNTR into Rt
```

To write this register in AArch32 state (CLUSTERPMXEVCNTR) from a general-purpose register, use the MCR (or MCR2) instruction with the following syntax:

```
MCR p15, 0, <Rt>, c15, c6, 2; Write Rt into CLUSTERPMXEVCNTR
```

Accessibility

This register is accessible in software as follows:

<systemreg>	Control			Accessibility			
	E2H	TGE	NS	EL0	EL1	EL2	EL3
CLUSTERPMXEVCNTR	x	x	0	-	RW	n/a	RW
CLUSTERPMXEVCNTR	x	0	1	-	RW	RW	RW
CLUSTERPMXEVCNTR	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

This register is write accessible in EL1 if ACTLR_EL3[12] is 1 and ACTLR_EL2[12] is 1, or ACTLR_EL3[12] is 1 and SCR.NS is 0. If write access is not possible, then Trap to the lowest Exception level that denied access (EL2 or EL3).

Related information

CLUSTERPMEVCNTR<n>, Cluster Event Counter Register on page 184

13.15 CLUSTERPMMDCR, Cluster Monitor Debug Configuration Register

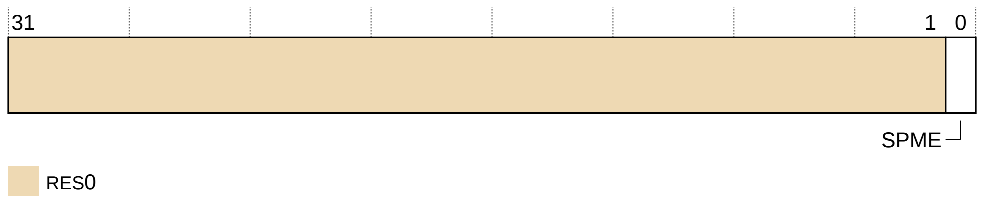
The CLUSTERPMMDCR register defines which common architectural and common microarchitectural feature events are implemented.

Bit field descriptions

This description applies to both the AArch32 (CLUSTERPMMDCR) and AArch64 (CLUSTERPMMDCR_EL3) registers.

CLUSTERPMMDCR is a 32-bit register, and is part of the PMU registers.

Figure 13-12: CLUSTERPMMDCR bit assignments



RES0, [31:1]

RES0	Reserved.

SPME, [0]

Secure Performance Monitors enable. This allows the counting of Secure events. The possible values are:

0	Counting of secure events prohibited.
1	Counting of secure events allowed.

This bit resets to zero.

Configurations

The AArch32 CLUSTERPMMDCR register is architecturally mapped to the AArch64 CLUSTERPMMDCR_EL3 register.

Usage constraints

Accessing the CLUSTERPMMDCR

In AArch64 state (CLUSTERPMMDCR_EL1):

To read this register in AArch64 state (CLUSTERPMMDCR_EL1) into a general-purpose register, use the MRS instruction with the following syntax:

```
MRS <Xt>, S3_6_C15_C6_3; Read CLUSTERPMMDCR_EL1 into Xt
```

To write this register in AArch64 state (CLUSTERPMMDCR_EL1) from a general-purpose register, use the MSR instruction with the following syntax:

```
MSR S3_6_C15_C6_3, <Xt>; Write Xt into CLUSTERPMMDCR_EL1
```

In AArch32 state (CLUSTERPMMDCR):

To read this register in AArch32 state (CLUSTERPMMDCR) into a general-purpose register, use the MRC (or MRC2) instruction with the following syntax:

```
MRC p15, 6, <Rt>, c15, c6, 3; Read CLUSTERPMMDCR into Rt
```

To write this register in AArch32 state (CLUSTERPMMDCR) from a general-purpose register, use the MCR (or MCR2) instruction with the following syntax:

```
MCR p15, 6, <Rt>, c15, c6, 3; Write Rt into CLUSTERPMMDCR
```

Accessibility

This register is accessible in software as follows:

<systemreg>	Control			Accessibility			
	E2H	TGE	NS	EL0	EL1	EL2	EL3
CLUSTERPMMDCR	x	x	0	-	-	n/a	RW
CLUSTERPMMDCR	x	0	1	-	-	-	RW

<systemreg>	Control			Accessibility			
	E2H	TGE	NS	EL0	EL1	EL2	EL3
CLUSTERPMMDCR	x	1	1	-	n/a	-	RW

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

Traps and enables

This register is only accessible from EL3.

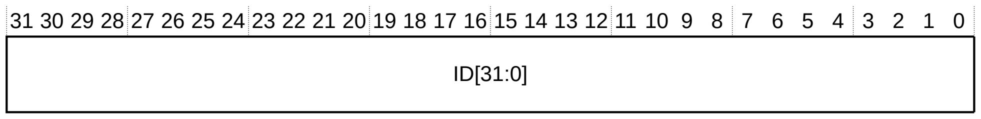
13.16 CLUSTERPMCEID0, Cluster Common Event Identification Register 0

The CLUSTERPMCEID0 register defines which common architectural and common microarchitectural feature events are implemented. This register is read-only write-ignores RO (WI).

Bit field descriptions

This description applies to both the AArch32 (CLUSTERPMCEID0) and AArch64 (CLUSTERPMCEID0_EL1) registers.

Figure 13-13: CLUSTERPMCEID0 bit assignments



See the PMCEID0 register description in the accompanying core for more information.

CE[31:0], [31:0]

Common architectural and microarchitectural feature events that can be counted by the PMU event counters.

The following table shows the CLUSTERPMCEID0 bit assignments with event implemented or not implemented when the associated bit is set to 1 or 0. See the *Arm® Architecture Reference Manual for A-profile architecture* for more information about these events.

Table 13-16: PMU events

Bit	Event number	Event mnemonic	Description
[30]	`0x1E`	CHAIN	Chain. For odd-numbered counters, counts when for each overflow of the preceding even-numbered counter. For even-numbered counters, does not count: **1** This event is implemented.

Bit	Event number	Event mnemonic	Description
[29]	`0x1D`	BUS_CYCLES	Bus cycle: **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.
[17]	`0x11`	CPU_CYCLES	Cycle: **1** This event is implemented.

Configurations

The AArch32 CLUSTERPMCEID0 register is architecturally mapped to the AArch64 CLUSTERPMCEID0_EL1 register.

Usage constraints

Accessing the CLUSTERPMCEID0

In AArch64 state (CLUSTERPMCEID0_EL1):

To read this register in AArch64 state (CLUSTERPMCEID0_EL1) into a general-purpose register, use the MRS instruction with the following syntax:

```
MRS <Xt>, S3_0_C15_C6_4; Read CLUSTERPMCEID0_EL1 into Xt
```

To write this register in AArch64 state (CLUSTERPMCEID0_EL1) from a general-purpose register, use the MSR instruction with the following syntax:

```
MSR S3_0_C15_C6_4, <Xt>; Write Xt into CLUSTERPMCEID0_EL1
```

In AArch32 state (CLUSTERPMCEID0):

To read this register in AArch32 state (CLUSTERPMCEID0) into a general-purpose register, use the MRC (or MRC2) instruction with the following syntax:

```
MRC p15, 0, <Rt>, c15, c6, 4; Read CLUSTERPMCEID0 into Rt
```

To write this register in AArch32 state (CLUSTERPMCEID0) from a general-purpose register, use the MCR (or MCR2) instruction with the following syntax:

```
MCR p15, 0, <Rt>, c15, c6, 4; Write Rt into CLUSTERPMCEID0
```

Accessibility

This register is accessible in software as follows:

<systemreg>	Control			Accessibility			
	E2H	TGE	NS	EL0	EL1	EL2	EL3
CLUSTERPMCEID0	x	x	0	-	RO/WI	n/a	RO/WI
CLUSTERPMCEID0	x	0	1	-	RO/WI	RO/WI	RO/WI
CLUSTERPMCEID0	x	1	1	-	n/a	RO/WI	RO/WI

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

Traps and enables

There are no traps or enables affecting this register.

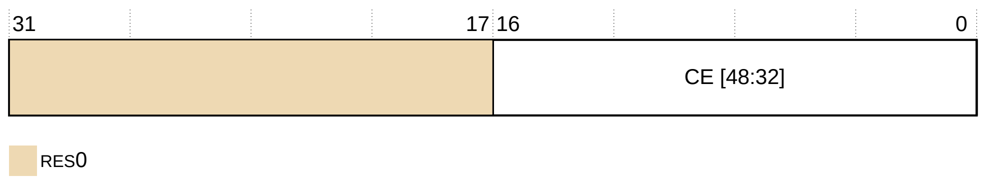
13.17 CLUSTERPMCEID1, Cluster Common Event Identification Register 1

The CLUSTERPMCEID1 register defines which common architectural and common microarchitectural feature events are implemented. This register is read-only write-ignores RO (WI).

Bit field descriptions

This description applies to both the AArch32 (CLUSTERPMCEID1) and AArch64 (CLUSTERPMCEID1_EL1) registers.

Figure 13-14: CLUSTERPMCEID1 bit assignments



See the PMCEID1 register description in the accompanying core for more information.

RES0, [31:17]

RES0	Reserved.

CE[48:32], [16:0]

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 13-18: PMU common event

Bit	Event number	Event mnemonic	Description
[12]	`0x2C`	L3D_CACHE_WB	Attributable Level 3 unified cache writeback. **1** This event is implemented.
[11]	`0x2B`	L3D_CACHE	Attributable Level 3 unified cache access. **1** This event is implemented.
[10]	`0x2A`	L3D_CACHE_REFILL	Attributable Level 3 unified cache refill. **1** This event is implemented.
[9]	`0x29`	L3D_CACHE_ALLOCATE	Attributable Level 3 unified cache allocation without refill. **1** This event is implemented.

Configurations

The AArch32 CLUSTERPMCEID1 register is architecturally mapped to the AArch64 CLUSTERPMCEID1_EL1 register.

Usage constraints

Accessing the CLUSTERPMCEID1

In AArch64 state (CLUSTERPMCEID1_EL1):

To read this register in AArch64 state (CLUSTERPMCEID1_EL1) into a general-purpose register, use the MRS instruction with the following syntax:

```
MRS <Xt>, S3_0_C15_C6_5; Read CLUSTERPMCEID1_EL1 into Xt
```

To write this register in AArch64 state (CLUSTERPMCEID1_EL1) from a general-purpose register, use the MSR instruction with the following syntax:

```
MSR S3_0_C15_C6_5, <Xt>; Write Xt into CLUSTERPMCEID1_EL1
```

In AArch32 state (CLUSTERPMCEID1):

To read this register in AArch32 state (CLUSTERPMCEID1) into a general-purpose register, use the MRC (or MRC2) instruction with the following syntax:

```
MRC p15, 0, <Rt>, c15, c6, 5; Read CLUSTERPMCEID1 into Rt
```

To write this register in AArch32 state (CLUSTERPMCEID1) from a general-purpose register, use the MCR (or MCR2) instruction with the following syntax:

```
MCR p15, 0, <Rt>, c15, c6, 5; Write Rt into CLUSTERPMCEID1
```

Accessibility

This register is accessible in software as follows:

<systemreg>	Control			Accessibility			
	E2H	TGE	NS	EL0	EL1	EL2	EL3
CLUSTERPMCEID1	x	x	0	-	RO/WI	n/a	RO/WI
CLUSTERPMCEID1	x	0	1	-	RO/WI	RO/WI	RO/WI
CLUSTERPMCEID1	x	1	1	-	n/a	RO/WI	RO/WI

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

Traps and enables

There are no traps or enables affecting this register.

13.18 CLUSTERCLAIMSET, Cluster Claim Tag Set Register

The CLUSTERCLAIMSET register provides various bits that can be separately set to indicate whether functionality is in use by a debug agent. All debug agents must implement a common protocol to use these bits.

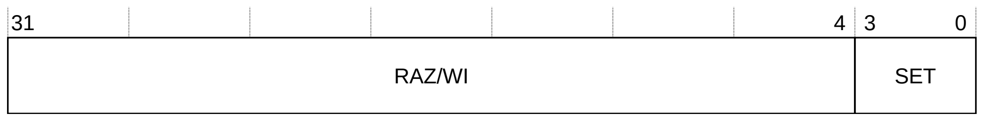
Bit field descriptions

For examples on how these bits can be used, see the CLAIMSET register description in the *Arm® CoreSight™ Architecture Specification v3.0*.

This description applies to both the AArch32 (CLUSTERCLAIMSET) and AArch64 (CLUSTERCLAIMSET_EL1) registers.

CLUSTERCLAIMSET is a 32-bit register, and is part of the PMU registers.

Figure 13-15: CLUSTERCLAIMSET bit assignments



RAZ/WI, [31:4]

Read-As-Zero, Writes Ignored.

SET, bits [3:0]

Each bit in this field is a SET bit. Permitted values of SET[n] are:

Write 0	No effect.
Write 1	Set the claim tag bit for bit[n].

Read 0	The claim tag that is represented by bit[n] is not implemented.
Read 1	The claim tag that is represented by bit[n] is implemented.

Configurations

The AArch32 CLUSTERCLAIMSET register is architecturally mapped to the AArch64 CLUSTERCLAIMSET_EL1 register.

Usage constraints

Accessing the CLUSTERCLAIMSET

In AArch64 state (CLUSTERCLAIMSET_EL1):

To read this register in AArch64 state (CLUSTERCLAIMSET_EL1) into a general-purpose register, use the MRS instruction with the following syntax:

```
MRS <Xt>, S3_0_C15_C6_6; Read CLUSTERCLAIMSET_EL1 into Xt
```

To write this register in AArch64 state (CLUSTERCLAIMSET_EL1) from a general-purpose register, use the MSR instruction with the following syntax:

```
MSR S3_0_C15_C6_6, <Xt>; Write Xt into CLUSTERCLAIMSET_EL1
```

In AArch32 state (CLUSTERCLAIMSET):

To read this register in AArch32 state (CLUSTERCLAIMSET) into a general-purpose register, use the MRC (or MRC2) instruction with the following syntax:

```
MRC p15, 0, <Rt>, c15, c6, 6; Read CLUSTERCLAIMSET into Rt
```

To write this register in AArch32 state (CLUSTERCLAIMSET) from a general-purpose register, use the MCR (or MCR2) instruction with the following syntax:

```
MCR p15, 0, <Rt>, c15, c6, 6; Write Rt into CLUSTERCLAIMSET
```

Accessibility

This register is accessible in software as follows:

<systemreg>	Control			Accessibility			
	E2H	TGE	NS	EL0	EL1	EL2	EL3
CLUSTERCLAIMSET	x	x	0	-	RW	n/a	RW
CLUSTERCLAIMSET	x	0	1	-	RW	RW	RW
CLUSTERCLAIMSET	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

This register is write accessible in EL1 if ACTLR_EL3[12] is 1 and ACTLR_EL2[12] is 1, or ACTLR_EL3[12] is 1 and SCR.NS is 0. If write access is not possible, then Trap to the lowest Exception level that denied access (EL2 or EL3).

13.19 CLUSTERCLAIMCLR, Cluster Claim Tag Clear Register

The CLUSTERCLAIMCLR register provides various bits that can be separately cleared to indicate whether functionality is in use by a debug agent. All debug agents must implement a common protocol to use these bits.

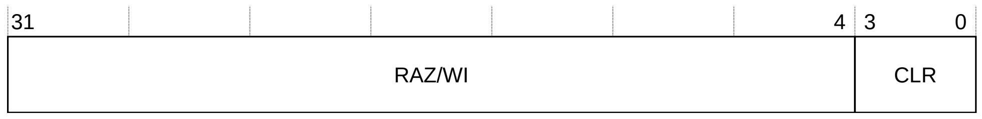
Bit field descriptions

For examples on how these bits can be used, see *Arm® CoreSight™ Architecture Specification v3.0*.

This description applies to both the AArch32 (CLUSTERCLAIMCLR) and AArch64 (CLUSTERCLAIMCLR_EL1) registers.

CLUSTERCLAIMCLR is a 32-bit register, and is part of the PMU registers.

Figure 13-16: CLUSTERCLAIMCLR bit assignments



RAZ/WI, [31:4]

Read-As-Zero, Writes Ignored.

CLR, bits [3:0]

Each bit in this field is a SET bit. Permitted values of SET[n] are:

Write 0	No effect.
Write 1	Set the claim tag bit for bit[n].
Read 0	The claim tag that is represented by bit[n] is not implemented.
Read 1	The claim tag that is represented by bit[n] is implemented.

Configurations

The AArch32 CLUSTERCLAIMCLR register is architecturally mapped to the AArch64 CLUSTERCLAIMCLR_EL1 register.

Usage constraints

Accessing the CLUSTERCLAIMCLR

In AArch64 state (CLUSTERCLAIMCLR_EL1):

To read this register in AArch64 state (CLUSTERCLAIMCLR_EL1) into a general-purpose register, use the MRS instruction with the following syntax:

```
MRS <Xt>, S3_0_C15_C6_7; Read CLUSTERCLAIMCLR_EL1 into Xt
```

To write this register in AArch64 state (CLUSTERCLAIMCLR_EL1) from a general-purpose register, use the MSR instruction with the following syntax:

```
MSR S3_0_C15_C6_7, <Xt>; Write Xt into CLUSTERCLAIMCLR_EL1
```

In AArch32 state (CLUSTERCLAIMCLR):

To read this register in AArch32 state (CLUSTERCLAIMCLR) into a general-purpose register, use the MRC (or MRC2) instruction with the following syntax:

```
MRC p15, 0, <Rt>, c15, c6, 7; Read CLUSTERCLAIMCLR into Rt
```

To write this register in AArch32 state (CLUSTERCLAIMCLR) from a general-purpose register, use the MCR (or MCR2) instruction with the following syntax:

```
MCR p15, 0, <Rt>, c15, c6, 7; Write Rt into CLUSTERCLAIMCLR
```

Accessibility

This register is accessible in software as follows:

<systemreg>	Control			Accessibility			
	E2H	TGE	NS	EL0	EL1	EL2	EL3
CLUSTERCLAIMCLR	x	x	0	-	RW	n/a	RW
CLUSTERCLAIMCLR	x	0	1	-	RW	RW	RW
CLUSTERCLAIMCLR	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

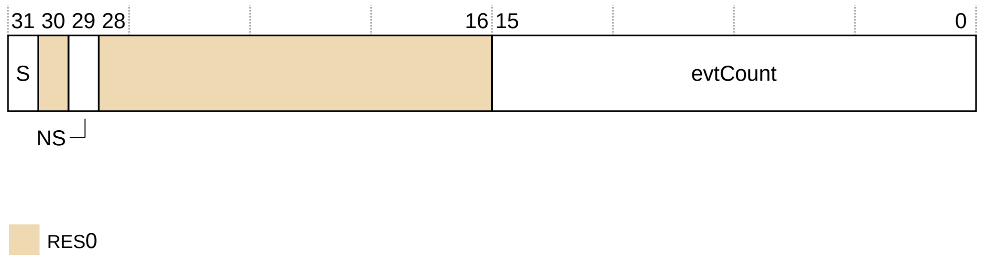
This register is write accessible in EL1 if ACTLR_EL3[12] is 1 and ACTLR_EL2[12] is 1, or ACTLR_EL3[12] is 1 and SCR.NS is 0. If write access is not possible, then Trap to the lowest Exception level that denied access (EL2 or EL3).

13.20 CLUSTERPMEVTYPER<n>, Cluster Event Type Register

The CLUSTERPMEVTYPER<n> register configures event counter n to count the specified event and its associated security filtering.

Bit field descriptions

Figure 13-17: CLUSTERPMEVTYPER<n> bit assignments



S, [31]

Secure events filtering bit. Controls counting of events that are generated by Secure transactions. The possible values are:

`0`	Count Secure events.
`1`	Do not count Secure events.

RES0, [30]

RES0	Reserved.

NS, [29]

Non-secure events filtering bit. Controls counting of events that are generated by Non-secure transactions. The possible values are:

`NS == S`	If the value of this bit equals the value of S,[31] bit then count Non-secure events.
`NS != S`	If the value of this bit does not equal the value of S,[31] bit then do not count Non-secure events.

RES0, [28:16]

RES0	Reserved.

evtCount, [15:0]

Event number. For the list of implemented events, see 15.3 PMU events - DSU R3 and R3 ADP on page 193.

If evtCount is programmed to an event that is not implemented, no events are counted.

Usage constraints

Accessing the CLUSTERPMEVTYPER<n> register

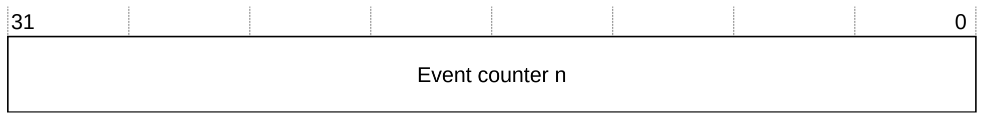
This register is accessible using the CLUSTERPMXEVTYPER register. It is not directly accessible as a system or CP15 register.

13.21 CLUSTERPMEVCNTR<n>, Cluster Event Counter Register

The CLUSTERPMEVCNTR<n> register holds the count for event counter n.

Bit field descriptions

Figure 13-18: CLUSTERPMEVCNTR<n> bit assignments



Bits, [31:0]

Event counter n. Value of event counter n, where n is the number of this register.

Usage constraints

Accessing the CLUSTERPMEVCNTR<n> register

This register is accessible using the CLUSTERPMXEVTYPER register. It is not directly accessible as a system or CP15 register.

14. Debug

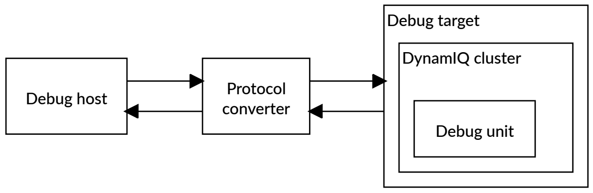
This chapter describes the debug features of the DSU and the associated DebugBlock component.

14.1 About debug methods

The DSU along with its associated cores is part of a debug system that supports both self-hosted and external debug.

The following figure shows a typical external debug system.

Figure 14-1: 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 DSU.

Debug unit

Helps debugging software that is running on the core:

- DSU and external hardware based around 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 PE.

For self-hosted debug, the debug target runs additional debug monitor software that runs on the core in the cluster. This way, it does not require expensive interface hardware to connect a second host computer.

14.2 Terminology

The debug system supports both single and multi-threaded cores.

The Arm architecture allows for cores to be single, or multi-threaded. A *Processing Element* (PE) performs a thread of execution. A single-threaded core has one PE and a multi-threaded core has two or more PEs. Because the debugging system allows individual threads to be debugged, the term PE is used throughout this chapter. Where a reference to a core is made, the core can be a single, or multi-threaded core.

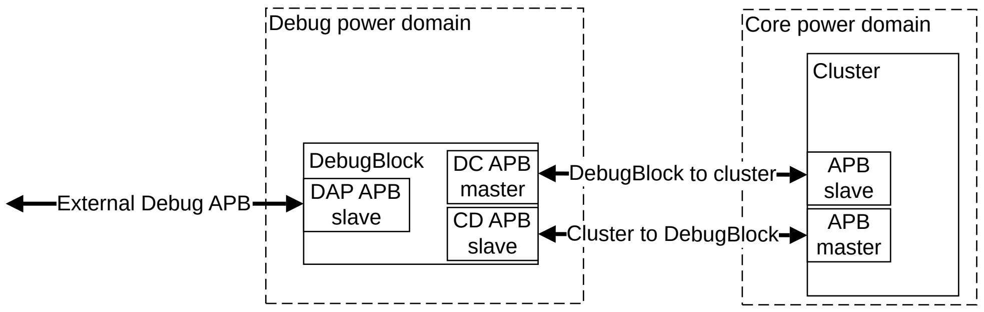
14.3 About the DebugBlock

The DebugBlock combines the functions, registers, and interfaces that are required for debug over powerdown.

The DebugBlock is provided as a separate component to allow implementation in a separate power domain from the cluster. Having a separate debug power domain allows the connection to a debugger be maintained while the cores and cluster are powered down.

The following diagram shows how the DebugBlock is connected to the cluster.

Figure 14-2: Debug APB connections



The DebugBlock has three APB interfaces:

External Debug APB (DAP APB)

An APB slave interface, allowing communication with an external debugger, for example through a CoreSight *Debug Access Port* (DAP).

All debug register read and write requests from an external debugger are received on this bus.

DebugBlock to cluster (DC APB)

An APB master interface that is connected to the cluster. It sends all debug register read and write requests to the cluster.

CTI output trigger events are sent to the cluster as trigger requests on this bus.

Cluster to DebugBlock (CD APB)

An APB slave interface that is connected to the cluster. It receives CTI input trigger event requests from the cluster.

Debug register reads and writes

The DebugBlock holds all the debug registers that are implemented in the Debug power domain. Registers implemented in the Debug power domain are specified in the *Arm® Architecture Reference Manual for A-profile architecture*.

Accesses through the DAP APB interface to Debug domain registers are handled internally by the DebugBlock. Accesses through the DAP APB interface to Core power domain registers are passed on to the cluster through the DC APB interface.

CTI trigger events

Trigger events are transferred between the DebugBlock and cluster through the CD APB and DC APB interfaces.

Input trigger events

Input trigger events are sent from the cluster to the CTIs through the CD APB as write transactions.

Output trigger events

Output trigger events are sent from the CTIs to the cluster through the DC APB as write transactions.

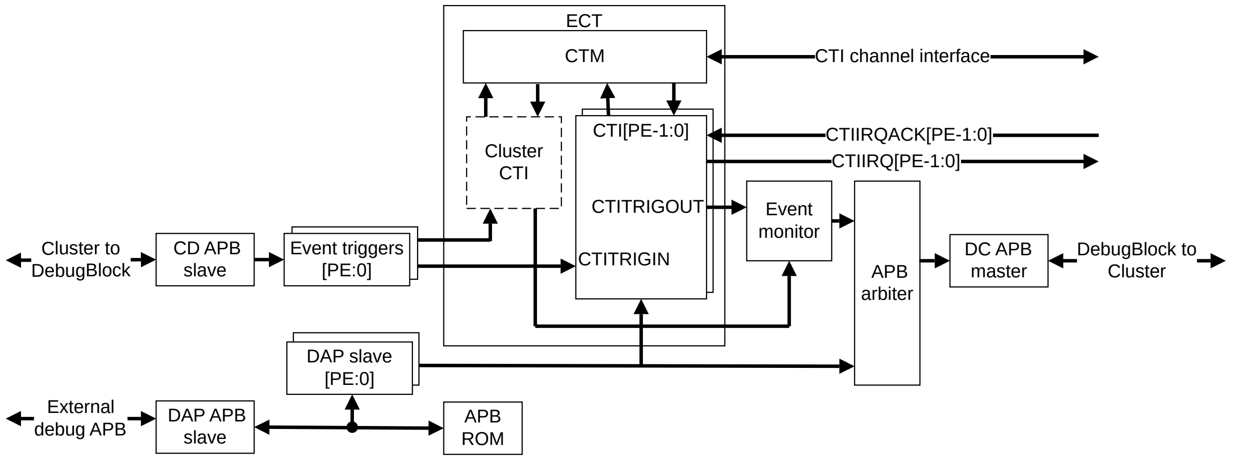
DebugBlock power states

The DebugBlock supports two power modes: ON and OFF. These power modes are controlled using the power Q-Channel interface, which due to an erratum does not function correctly. Therefore, Arm recommends that the DebugBlock power Q-Channel is not used and that PWRQREQn is tied HIGH. To power down the DebugBlock, Arm recommends that the DebugBlock is put in reset before powering down. This causes any transactions, on the external Debug APB interface, that have not completed to complete with a SLVERR.

14.4 DebugBlock components

The components are:

Figure 14-3: DebugBlock block diagram



Note

The CTIs shown in the diagram includes both the CTIs attached to each of the PEs [0:PE-1] and the cluster CTI. The cluster CTI is present only when the cluster ELA is present.

ECT

The DebugBlock implements the *Embedded Cross Trigger* (ECT).

APB ROM

The APB ROM table holds the address decoding for each debug component in the DebugBlock and the cluster. The APB ROM table complies with the *Arm® CoreSight™ Architecture Specification v3.0*. Both v7 and v8 debug address maps are supported.

Event monitor

The event monitor converts changes in CTI output triggers to APB write transactions.

Event triggers

The event triggers convert APB write transactions to CTI input triggers.

APB arbiter

The DC APB transfers both register accesses and CTI output trigger events. The APB arbiter multiplexes the two sources of transactions.

DAP slave

The DAP slave holds copies of registers in the debug power domain.

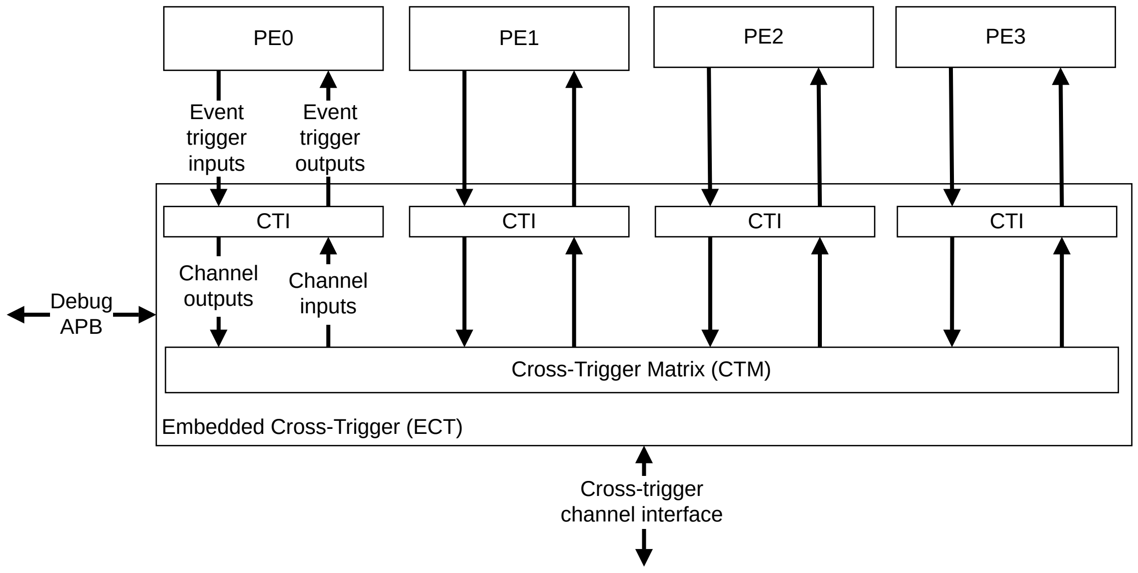
14.5 About the Embedded Cross Trigger

The *Embedded Cross Trigger* (ECT) allows debug events to be sent between PEs.

The ECT provides a *Cross Trigger Interface* (CTI) for each PE in the cluster. The CTIs are interconnected through a *Cross Trigger Matrix* (CTM) to send debug and trace events between PEs.

The following diagram shows a conceptual view of the trigger event inputs and outputs between the PEs and ECT.

Figure 14-4: Embedded Cross Trigger concept



The CTIs selectively send trigger events to the CTM on their respective channel outputs. The CTIs receive trigger events from the CTM on their channel inputs.

Trigger events are transferred between CTIs over the channel interface. The CTM connects the channel interface to the channel inputs and channel outputs of the CTIs.

External interfaces

The external cross-trigger channel interface, from the CTM, allows cross-triggering between SoC external devices.

The Debug APB provides access to the CTI registers. This allows an external debugger to configure the trigger event routing, and send events to PEs, for example, to put a PE into Debug state.

CTI registers

Registers in the CTI:

- Control the mapping of the input trigger events to channel outputs.

- Control the mapping of the channel inputs to output trigger events.
- Capture the state of input and output trigger events.
- Set, clear, or pulse output trigger events.

14.5.1 Supported debug and trace trigger events

The CTIs each have nine input and output trigger events that are mapped onto the debug and trace events in the PEs and ELAs.

The debug and trace trigger events from the CTI to the PE are:

Debug request trigger event
A trigger event sent from the CTI to the PE to force the PE into Debug state.

Restart request trigger event
A trigger event sent from the CTI to the PE to request the PE to exit Debug state.

Generic CTI interrupt trigger event
A trigger event sent from the CTI to the GIC.

ETM trace input trigger events
Four trigger events sent from the CTI to the ETM trace in the PE.

ELA input trigger events
Two trigger events sent from the CTI to the ELA attached to the PE.

The debug and trace events from the PE to the CTI are:

Cross-halt trigger event
A trigger event sent from the PE to the CTI when the PE enters Debug state.

Performance Monitors overflow trigger event
A trigger event sent from the PE to the CTI when a PMU event counter overflows.

ETM trace output trigger events
Four trigger events sent from the ETM in the PE to the CTI.

ELA output trigger events
Two trigger events sent from the ELA (attached to the PE) to the CTI.

Profiling sample trigger event
A trigger event sent from the PE to the CTI when a profiling sample is written out.

The cluster CTI has two input and output trigger events that are mapped onto the trigger events in the cluster ELA. The trigger events from the cluster CTI to the cluster ELA are:

Cluster ELA input trigger events
Two trigger events sent from the cluster CTI to the cluster ELA.

The trigger events from the cluster ELA to the cluster CTI are:

Cluster ELA output trigger events

Two trigger events from the cluster ELA to the cluster CTI.

14.6 CTI triggers

Events are mapped onto CTI input and output triggers. All PEs in the cluster have the same mapping.

PE CTI input trigger events

The following table shows how events are mapped onto PE CTI input triggers.

Table 14-1: Allocation of PE CTI trigger inputs

Trigger number	Source	Destination	Type	Event description
0	PE	CTI	Pulse	Cross-halt trigger event
1				Performance Monitors Overflow trigger event
2	PE	CTI	Pulse	Profiling sample trigger event
3	-	-	-	Reserved
4-7	ETM	CTI	Pulse	ETM Trace Output trigger events
8-9	ELA	CTI		ELA CTTRIGOUT[1:0] trigger events

PE CTI output trigger events

The following table shows how events are mapped onto PE CTI output triggers.

Table 14-2: Allocation of PE CTI trigger outputs

Trigger number	Source	Destination	Type	Event description
0	CTI	PE	Level	Debug Request trigger event
1			Pulse	Restart Request trigger event
2	CTI	GIC	Pulse	Generic CTI Interrupt trigger event
3	-	-	-	Reserved
4-7	CTI	ETM	Pulse	Generic Trace External Input trigger events
8-9	CTI	ELA	Pulse	ELA CTTRIGIN[1:0] trigger events

Allocation of cluster CTI trigger inputs

The following table shows how events are mapped onto the cluster CTI input triggers.

Table 14-3: Allocation of cluster CTI trigger inputs

Trigger number	Source	Destination	Type	Event description
0-7	-	-	-	Reserved
8-9	Cluster ELA	Cluster CTI	Pulse	Cluster ELA CTTRIGOUT[1:0]

Allocation of cluster CTI trigger outputs

The following table shows how events are mapped onto the cluster CTI output triggers.

Table 14-4: Allocation of cluster CTI trigger inputs

Trigger number	Source	Destination	Type	Event description
0-7	-	-	-	Reserved
8-9	Cluster CTI	Cluster ELA	Pulse	Cluster ELA CTTRIGIN[1:0]

15. Performance Monitoring Unit

This chapter describes the *Performance Monitoring Unit* (PMU).

15.1 About the PMU

The DynamIQ™ Shared Unit includes performance monitors that enable you to gather various statistics on the operation of the memory of the cluster during runtime. These provide useful information about the behavior of the cluster 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 cluster. The absolute counts that are recorded might vary because of pipeline effects. This has negligible effect except in cases where the counters are enabled for a very short time.

15.2 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

You can program the PMU registers using the system registers.

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 DynamIQ™ Shared Unit supports access to the performance monitor registers from the internal system register interface.

Related information

PMU registers on page 149

15.3 PMU events

The following table shows the events that are generated and the numbers that the PMU uses to reference the events.

Table 15-1: PMU events

PMU event number	Event mnemonic	Event description
0x0011	CYCLES	Cycle counter.
0x0019	BUS_ACCESS	Bus access counter. Counts every beat of data that is transferred over the data channels between the SCU and the interconnect. This event counts the sum of BUS_ACCESS_RD and BUS_ACCESS_WR.
0x001A	MEMORY_ERROR	Local memory error counter. Counts every Correctable or Uncorrectable memory error (ECC or parity) in the protected RAMs.
0x001D	BUS_CYCLES	ACE or CHI bus cycle counter.
0x001E	CHAIN	Odd performance counter chain mode.
0x0029	L3D_CACHE_ALLOCATE	Level 3 unified cache allocation without refill counter. Counts every full cache line write into the L3 cache which does not cause a linefill.
0x002A	L3D_CACHE_REFILL	Level 3 unified cache refill counter. Counts every cacheable read transaction issued to the interconnect. This event counts the sum of L3D_CACHE_REFILL_RD and L3D_CACHE_REFILL_WR.
0x002B	L3D_CACHE	Level 3 unified cache access counter. Counts every cacheable read or write transaction issued to the SCU. This event counts the sum of L3D_CACHE_RD and L3D_CACHE_WR.
0x002C	L3D_CACHE_WB	Level 3 unified cache write-back counter. Counts every write-back from the L3 cache.
0x0060	BUS_ACCESS_RD	Bus access, read counter. Counts every beat of data transferred over the read data channel between the SCU and the interconnect.
0x0061	BUS_ACCESS_WR	Bus access, write counter. Counts every beat of data transferred over the write data channel between the SCU and the interconnect.
0x0062	BUS_ACCESS_SHARED	Bus access, shared counter. Counts every beat of shared data transferred over the data channels between the SCU and the interconnect.
0x0063	BUS_ACCESS_NOT_SHARED	Bus access, not shared counter. Counts every beat of not shared data transferred over the write data channel between the SCU and the interconnect.
0x0064	BUS_ACCESS_NORMAL	Bus access, normal counter. Counts every beat of normal data transferred over the write data channel between the SCU and the interconnect.
0x0065	BUS_ACCESS_PERIPH	Bus access, periph counter. Counts every beat of device data transferred over the write data channel between the SCU and the interconnect.
0x00A0	L3D_CACHE_RD	Level 3 unified cache access, read counter. Counts every cacheable read transaction that is issued to the SCU. Prefetches and stashes are not counted.
0x00A1	L3D_CACHE_WR	Level 3 unified cache access, write counter. Counts every cacheable write transaction issued to the SCU.
0x00A2	L3D_CACHE_REFILL_RD	Level 3 unified cache refill, read counter. Counts every cacheable read transaction issued to the interconnect caused by a read transaction. Prefetches and stashes are not counted.

PMU event number	Event mnemonic	Event description
`0x00A3`	L3D_CACHE_REFILL_WR	Level 3 unified cache refill, write counter. Counts every cacheable read transaction issued to the interconnect caused by a write transaction.
`0x0119`	ACP_ACCESS	ACP access counter. Counts every beat of data transferred over the data channels between the SCU and the accelerated coherency port. This event counts the sum of ACP_ACCESS_RD and ACP_ACCESS_WR.
`0x011D`	ACP_CYCLES	ACP cycle counter.
`0x0160`	ACP_ACCESS_RD	ACP access, read counter. Counts every beat of data transferred over the read data channel between the SCU and the peripheral port.
`0x0161`	ACP_ACCESS_WR	ACP access, write counter. Counts every beat of data transferred over the write data channel between the SCU and the peripheral port.
`0x0219`	PP_ACCESS	Peripheral port access counter. Counts every beat of data transferred over the data channels between the SCU and the peripheral port. This event counts the sum of PP_ACCESS_RD and PP_ACCESS_WR.
`0x021D`	PP_CYCLES	Peripheral port cycle counter.
`0x0260`	PP_ACCESS_RD	Peripheral port access, read counter. Counts every beat of data transferred over the read data channel between the SCU and the peripheral port.
`0x0261`	PP_ACCESS_WR	Peripheral port access, write counter. Counts every beat of data transferred over the write data channel between the SCU and the peripheral port.
`0x00C0`	SCU_SNP_ACCESS	SNP access counter. Counts every external snoop request.
`0x00C1`	SCU_SNP_EVICT	SNP evictions counter. Counts every invalidating external snoop request that causes an L3 cache eviction.
`0x00C2`	SCU_SNP_NO_CPU_SNP	SNP, no CPU snoop counter. Counts every external snoop request that completes without needing to snoop a core.
`0x0500`	SCU_PFTCH_CPU_ACCESS	Prefetch access, CPU counter. Counts every L3 prefetch transaction originating from a core.
`0x0501`	SCU_PFTCH_CPU_MISS	Prefetch data miss, CPU counter. Counts every L3 prefetch transaction originating from a core where data was read in from outside the cluster.
`0x0502`	SCU_PFTCH_CPU_HIT	Prefetch data hit, CPU counter. Counts every L3 prefetch transaction originating from a core where the L3 prefetch hit in the cluster.
`0x0503`	SCU_PFTCH_CPU_MATCH	Prefetch match, CPU counter. Counts every completed L3 prefetch transaction originating from a core that is matched by a compatible read request. This includes one caused by a L3 prefetch to the core, before the L3 prefetch times-out and is allocated into the L3 cache.
`0x0504`	SCU_PFTCH_CPU_KILL	Prefetch terminate, CPU counter. Counts every killed L3 prefetch transaction originating from a core that is terminated due to an incompatible match.
`0x0510`	SCU_STASH_ICN_ACCESS	Stash access, ICN counter. Counts every stash transaction originating from the interconnect.
`0x0511`	SCU_STASH_ICN_MISS	Stash data miss, ICN counter. Counts every stash transaction originating from the interconnect which utilizes a data pull, or is added to the stash queue and later issues a read.
`0x0512`	SCU_STASH_ICN_HIT	Stash data hit, ICN counter. Counts every non-invalidating stash transaction originating from the interconnect which hits in the cluster.
`0x0513`	SCU_STASH_ICN_MATCH	Stash match, ICN counter. Counts every completed stash transaction originating from the interconnect which is matched by a compatible read request. This includes one caused by a stash to the core, before the stash times out and is allocated into the L3 cache.
`0x0514`	SCU_STASH_ICN_KILL	Stash terminated, ICN counter. Counts every killed stash transaction originating from the interconnect that is terminated due to an incompatible match.
`0x00D0`	SCU_HZD_ADDRESS	Arbitration hazard, address counter. Counts every flush caused by an address hazard.

15.4 PMU interrupts

The DSU asserts the nCLUSTERPMUIRQ 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 a core.

16. Debug registers

This chapter describes the debug registers for the DSU.

16.1 Debug memory map

The debug memory map holds the base addresses for each debug component that is connected to the Debug APB.

Each component in the table requires 4KB, and uses the bottom 4KB of each 64KB region. The remaining 60KB of each region is reserved.

In the tables, individual *Processing Elements* (PE) are identified. For a single-threaded core, the PE number is the same as the core number. Only eight cores are supported.

The following table shows the address mapping for the DSU and the individual core debug APB components when configured for v8 Debug memory map. If an address range is not mapped to a component, it is indicated as reserved. For more information, see *Arm® Architecture Reference Manual for A-profile architecture*.

Table 16-1: Address mapping for APB components on 64KB pages

Address	Component
`0x000000`	ROM table
`0x010000`	PE0 Debug
`0x020000`	PE0 CTI
`0x030000`	PE0 PMU
`0x040000`	PE0 ETM
`0x050000-0x0BFFFF`	Reserved, RES0
`0x0C0000`	PE0 ELA
`0x0D0000`	Cluster ELA
`0x0E0000`	Cluster CTI
`0x0F0000`	PE0 Activity Monitor
`0x100000-0x10FFFF`	Reserved, RES0
`0x110000`	PE1 Debug
`0x120000`	PE1 CTI
`0x130000`	PE1 PMU
`0x140000`	PE1 ETM
`0x150000-0x1BFFFF`	Reserved, RES0
`0x1C0000`	PE1 ELA
`0x1D0000-0x1EFFFF`	Reserved, RES0
`0x1F0000`	PE1 Activity Monitor
`0x200000-0x20FFFF`	Reserved, RES0

Address	Component
`0x210000`	PE2 Debug
`0x220000`	PE2 CTI
`0x230000`	PE2 PMU
`0x240000`	PE2 ETM
`0x250000-0x2BFFFF`	Reserved, RES0
`0x2C0000`	PE2 ELA
`0x2D0000-0x2EFFFF`	Reserved, RES0
`0x2F0000`	PE2 Activity Monitor
`0x300000-0x30FFFF`	Reserved, RES0
`0x310000`	PE3 Debug
`0x320000`	PE3 CTI
`0x330000`	PE3 PMU
`0x340000`	PE3 ETM
`0x350000-0x3BFFFF`	Reserved, RES0
`0x3C0000`	PE3 ELA
`0x3D0000-0x3EFFFF`	Reserved, RES0
`0x3F0000`	PE3 Activity Monitor
`0x400000-0x40FFFF`	Reserved, RES0
`0x410000`	PE4 Debug
`0x420000`	PE4 CTI
`0x430000`	PE4 PMU
`0x440000`	PE4 ETM
`0x450000-0x4BFFFF`	Reserved, RES0
`0x4C0000`	PE4 ELA
`0x4D0000-0x4EFFFF`	Reserved, RES0
`0x4F0000`	PE4 Activity Monitor
`0x500000-0x50FFFF`	Reserved, RES0
`0x510000`	PE5 Debug
`0x520000`	PE5 CTI
`0x530000`	PE5 PMU
`0x540000`	PE5 ETM
`0x550000-0x5BFFFF`	Reserved, RES0
`0x5C0000`	PE5 ELA
`0x5D0000-0x5EFFFF`	Reserved, RES0
`0x5F0000`	PE5 Activity Monitor
`0x600000-0x60FFFF`	Reserved, RES0
`0x610000`	PE6 Debug
`0x620000`	PE6 CTI
`0x630000`	PE6 PMU
`0x640000`	PE6 ETM

Address	Component
`0x650000-0x6BFFFF`	Reserved, RES0
`0x6C0000`	PE6 ELA
`0x6D0000-0x6EFFFF`	Reserved, RES0
`0x6F0000`	PE6 Activity Monitor
`0x700000-0x70FFFF`	Reserved, RES0
`0x710000`	PE7 Debug
`0x720000`	PE7 CTI
`0x730000`	PE7 PMU
`0x740000`	PE7 ETM
`0x750000-0x7BFFFF`	Reserved, RES0
`0x7C0000`	PE7 ELA
`0x7D0000-0x7EFFFF`	Reserved, RES0
`0x7F0000`	PE7 Activity Monitor
`0x800000-0x80FFFF`	Reserved, RES0
`0x810000`	PE8 Debug
`0x820000`	PE8 CTI
`0x830000`	PE8 PMU
`0x840000`	PE8 ETM
`0x850000-0x8BFFFF`	Reserved, RES0
`0x8C0000`	PE8 ELA
`0x8D0000-0x8EFFFF`	Reserved, RES0
`0x8F0000`	PE8 Activity Monitor
`0x900000-0x90FFFF`	Reserved, RES0
`0x910000`	PE9 Debug
`0x920000`	PE9 CTI
`0x930000`	PE9 PMU
`0x940000`	PE9 ETM
`0x950000-0x9BFFFF`	Reserved, RES0
`0x9C0000`	PE9 ELA
`0x9D0000-0x9EFFFF`	Reserved, RES0
`0x9F0000`	PE9 Activity Monitor
`0xA00000-0xA0FFFF`	Reserved, RES0
`0xA10000`	PE10 Debug
`0xA20000`	PE10 CTI
`0xA30000`	PE10 PMU
`0xA40000`	PE10 ETM
`0xA50000-0xABFFFF`	Reserved, RES0
`0xAC0000`	PE10 ELA
`0xAD0000-0xAEFFFF`	Reserved, RES0
`0xAF0000`	PE10 Activity Monitor

Address	Component
`0xB00000-0xB0FFFF`	Reserved, RES0
`0xB10000`	PE11 Debug
`0xB20000`	PE11 CTI
`0xB30000`	PE11 PMU
`0xB40000`	PE11 ETM
`0xB50000-0xBBFFFF`	Reserved, RES0
`0xBC0000`	PE11 ELA
`0xBD0000-0xBEFFFF`	Reserved, RES0
`0xBF0000`	PE11 Activity Monitor
`0xC00000-0xC0FFFF`	Reserved, RES0
`0xC10000`	PE12 Debug
`0xC20000`	PE12 CTI
`0xC30000`	PE12 PMU
`0xC40000`	PE12 ETM
`0xC50000-0xCBFFFF`	Reserved, RES0
`0xCC0000`	PE12 ELA
`0xCD0000-0xCEFFFF`	Reserved, RES0
`0xCF0000`	PE12 Activity Monitor
`0xD00000-0xD0FFFF`	
`0xD10000`	PE13 Debug
`0xD20000`	PE13 CTI
`0xD30000`	PE13 PMU
`0xD40000`	PE13 ETM
`0xD50000-0xDBFFFF`	Reserved, RES0
`0xDC0000`	PE13 ELA
`0xDD0000-0xDEFFFF`	Reserved, RES0
`0xDF0000`	PE13 Activity Monitor
`0xE00000-0xE0FFFF`	Reserved, RES0
`0xE10000`	PE14 Debug
`0xE20000`	PE14 CTI
`0xE30000`	PE14 PMU
`0xE40000`	PE14 ETM
`0xE50000-0xEBFFFF`	Reserved, RES0
`0xEC0000`	PE14 ELA
`0xED0000-0xEEFFFF`	Reserved, RES0
`0xEF0000`	PE14 Activity Monitor
`0xF00000-0xF0FFFF`	Reserved, RES0
`0xF10000`	PE15 Debug
`0xF20000`	PE15 CTI
`0xF30000`	PE15 PMU

Address	Component
`0xF40000`	PE15 ETM
`0xF50000-0xFBFFFF`	Reserved, RES0
`0xFC0000`	PE15 ELA
`0xFD0000-0xFEFFFF`	Reserved, RES0
`0xFF0000`	PE15 Activity Monitor

The following table shows the address mapping for the DSU and the individual core debug APB components when configured for v7 Debug memory map. If an address range is not mapped to a component, it is indicated as reserved.

Table 16-2: Address mapping for APB components on 4KB pages

Address	Component
`0x000000`	ROM table
`0x001000`	Cluster ELA
`0x002000`	Cluster CTI
`0x003000` - `0x3FFFF`	Reserved
`0x004000`	PE0 Debug
`0x005000`	PE1 Debug
`0x006000`	PE2 Debug
`0x007000`	PE3 Debug
`0x008000`	PE0 ELA
`0x009000`	PE1 ELA
`0x00A000`	PE2 ELA
`0x00B000`	PE3 ELA
`0x00C000`	PE0 PMU
`0x00D000`	PE1 PMU
`0x00E000`	PE2 PMU
`0x00F000`	PE3 PMU
`0x010000-0x013FFF`	Reserved
`0x014000`	PE0 CTI
`0x015000`	PE1 CTI
`0x016000`	PE2 CTI
`0x017000`	PE3 CTI
0x018000	PE0 Activity Monitor
`0x019000`	PE1 Activity Monitor
`0x01A000`	PE2 Activity Monitor
`0x01B000`	PE3 Activity Monitor
`0x01C000`	PE0 ETM
`0x01D000`	PE1 ETM
`0x01E000`	PE2 ETM
`0x01F000`	PE3 ETM

Address	Component
`0x020000-0x023FFF`	Reserved
`0x024000`	PE4 Debug
`0x025000`	PE5 Debug
`0x026000`	PE6 Debug
`0x027000`	PE7 Debug
`0x028000`	PE4 ELA
`0x029000`	PE5 ELA
`0x02A000`	PE6 ELA
`0x02B000`	PE7 ELA
`0x02C000`	PE4 PMU
`0x02D000`	PE5 PMU
`0x02E000`	PE6 PMU
`0x02F000`	PE7 PMU
`0x030000-0x033FFF`	Reserved
`0x034000`	PE4 CTI
`0x035000`	PE5 CTI
`0x036000`	PE6 CTI
`0x037000`	PE7 CTI
`0x038000`	PE4 Activity Monitor
`0x039000`	PE5 Activity Monitor
`0x03A000`	PE6 Activity Monitor
`0x03B000`	PE7 Activity Monitor
`0x03C000`	PE4 ETM
`0x03D000`	PE5 ETM
`0x03E000`	PE6 ETM
`0x03F000`	PE7 ETM
`0x040000-0x043FFF`	Reserved
`0x044000`	PE8 Debug
`0x045000`	PE9 Debug
`0x046000`	PE10 Debug
`0x047000`	PE11 Debug
`0x048000`	PE8 ELA
`0x049000`	PE9 ELA
`0x04A000`	PE10 ELA
`0x04B000`	PE11 ELA
`0x04C000`	PE8 PMU
`0x04D000`	PE9 PMU
`0x04E000`	PE10 PMU
`0x04F000`	PE11 PMU
`0x050000-0x053FFF`	Reserved

Address	Component
`0x054000`	PE8 CTI
`0x055000`	PE9 CTI
`0x056000`	PE10 CTI
`0x057000`	PE11 CTI
`0x058000`	PE8 Activity Monitor
`0x059000`	PE9 Activity Monitor
`0x05A000`	PE10 Activity Monitor
`0x05B000`	PE11 Activity Monitor
`0x05C000`	PE8 ETM
`0x05D000`	PE9 ETM
`0x05E000`	PE10 ETM
`0x05F000`	PE11 ETM
`0x060000-0x063FFF`	Reserved
`0x064000`	PE12 Debug
`0x065000`	PE13 Debug
`0x066000`	PE14 Debug
`0x067000`	PE15 Debug
`0x068000`	PE12 ELA
`0x069000`	PE13 ELA
`0x06A000`	PE14 ELA
`0x06B000`	PE15 ELA
`0x06C000`	PE12 PMU
`0x06D000`	PE13 PMU
`0x06E000`	PE14 PMU
`0x06F000`	PE15 PMU
`0x070000-0x073FFF`	Reserved
`0x074000`	PE12 CTI
`0x075000`	PE13 CTI
`0x076000`	PE14 CTI
`0x077000`	PE15 CTI
`0x078000`	PE12 Activity Monitor
`0x079000`	PE13 Activity Monitor
`0x07A000`	PE14 Activity Monitor
`0x07B000`	PE15 Activity Monitor
`0x07C000`	PE12 ETM
`0x07D000`	PE13 ETM
`0x07E000`	PE14 ETM
`0x07F000`	PE15 ETM

16.2 CTI register summary

This section describes the CTI registers in the *DynamIQ™ Shared Unit* (DSU). These registers are accessed through the external debug interface.

The following table gives a summary of the CTI registers. For registers that are not described in this chapter, see the *Arm® Architecture Reference Manual for A-profile architecture*.

Table 16-3: CTI register summary

Offset	CTI	Description
`0x000`	CTICONTROL	CTI Control register
`0x004-0x00C`	-	Reserved
`0x010`	CTIINTACK	CTI Output Trigger Acknowledge Register
`0x014`	CTIAPPSET	CTI Application Trigger Set Register
`0x018`	CTIAPPCLEAR	CTI Application Trigger Clear Register
`0x01C`	CTIAPPPULSE	CTI Application Pulse Register
`0x020`	CTIINEN0	CTI Input Trigger to Output Channel Enable Registers
`0x024`	CTIINEN1	
`0x028`	CTIINEN2	
`0x02C`	CTIINEN3	
`0x030`	CTIINEN4	
`0x034`	CTIINEN5	
`0x038`	CTIINEN6	
`0x03C`	CTIINEN7	
`0x040`	CTIINEN8	
`0x044`	CTIINEN9	
`0x048-0x09C`	-	Reserved
`0x0A0`	CTIOUTEN0	CTI Input Channel to Output Trigger Enable Registers
`0x0A4`	CTIOUTEN1	
`0x0A8`	CTIOUTEN2	
`0x0AC`	CTIOUTEN3	
`0x0B0`	CTIOUTEN4	
`0x0B4`	CTIOUTEN5	
`0x0B8`	CTIOUTEN6	
`0x0BC`	CTIOUTEN7	
`0x0C0`	CTIOUTEN8	
`0x0C4`	CTIOUTEN9	
`0x0C8-0x12C`	-	Reserved
`0x130`	CTITRIGINSTATUS	CTI Trigger In Status Register
`0x134`	CTITRIGOUTSTATUS	CTI Trigger Out Status Register
`0x138`	CTICHINSTATUS	CTI Channel In Status Register
`0x13C`	CTICHOUTSTATUS	CTI Channel Out Status Register

Offset	CTI	Description
`0x140`	CTIGATE	CTI Channel Gate Enable Register
`0x144-0xF97`	-	Reserved
`0xFA0`	CTICLAIMSET	CTI Claim Tag Set Register
`0xFA4`	CTICLAIMCLR	CTI Claim Tag Clear Register
`0xFA8`	CTIDEVAFF0	16.9 CTIDEVAFF0, Cluster CTI Device Affinity register 0 on page 212 [10]
		CTI Device Affinity Register 0[11]
`0xFAC`	CTIDEVAFF1	CTI Device Affinity Register 1
`0xFB0-0xFB4`	-	Reserved
`0xFB8`	CTIAUTHSTATUS	CTI Authentication Status Register
`0xFBC`	CTIDEVARCH	CTI Device Architecture Register
`0xFC0`	CTIDEVID2	CTI Device Identification Register 2
`0xFC4`	CTIDEVID1	CTI Device Identification Register 1
`0xFC8`	CTIDEVID	16.10 CTIDEVID, CTI Device Identification Register on page 213
`0xFCC`	CTIDEVTYPE	CTI Device Type Register
`0xFD0`	CTIPIDR4	16.7 CTIPIDR4, CTI Peripheral Identification Register 4 on page 210
`0xFD4-0xFDC`	-	Reserved
`0xFE0`	CTIPIDR0	16.3 CTIPIDR0, CTI Peripheral Identification Register 0 on page 205
`0xFE4`	CTIPIDR1	16.4 CTIPIDR1, CTI Peripheral Identification Register 1 on page 206
`0xFE8`	CTIPIDR2	16.5 CTIPIDR2, CTI Peripheral Identification Register 2 on page 208
`0xFEC`	CTIPIDR3	16.6 CTIPIDR3, CTI Peripheral Identification Register 3 on page 209
`0xFF0`	CTICIDR0	CTI Component Identification Register 0
`0xFF4`	CTICIDR1	CTI Component Identification Register 1
`0xFF8`	CTICIDR2	CTI Component Identification Register 2
`0xFFC`	CTICIDR3	CTI Component Identification Register 3

16.3 CTIPIDR0, CTI Peripheral Identification Register 0

The CTIPIDR0 register provides information to identify a CTI component.

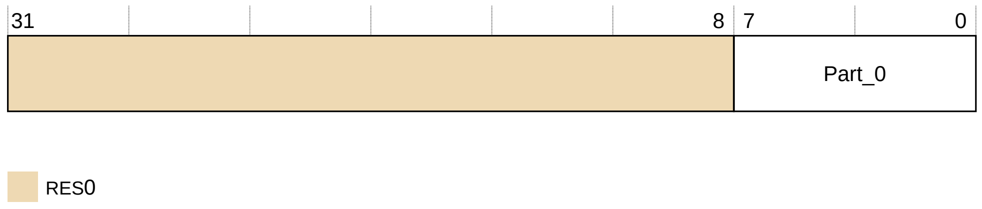
Bit field descriptions

CTIPIDR0 is a 32-bit register.

This register is Read Only.

[10] The CTIDEVAFF0 register for the cluster has a different format.
[11] The CTIDEVAFF0 register for the PEs is described in *Arm® Architecture Reference Manual for A-profile architecture*.

Figure 16-1: CTIPIDR0 bit assignments



RES0, [31:8]

RES0	Reserved.

Part_0, [7:0]

Least significant byte of the part number.

For the cluster, the value is:

`0xE3`	If v7 debug memory map is implemented.
`0xE4`	If v8 debug memory map is implemented.

For the PEs, the value is a copy of bits [11:4] of the Main ID Register (MIDR) of the corresponding PE.

Configurations

The CTIPIDR0 is in the Debug power domain and is optional to implement in the external register interface.

Usage constraints

Accessing the CTIPIDR0

The CTIPIDR0 can be accessed through the external debug interface with offset `0xFE0`.

Accessibility

The accessibility of the CTIPIDR0 by condition code is:

Default
RO

See 16.11 External register access permissions on page 214 for the condition codes.

16.4 CTIPIDR1, CTI Peripheral Identification Register 1

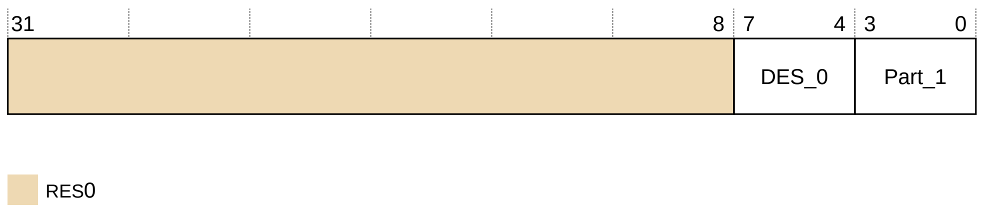
The CTIPIDR1 register provides information to identify a CTI component.

Bit field descriptions

CTIPIDR1 is a 32-bit register.

This register is Read Only.

Figure 16-2: CTIPIDR1 bit assignments



RES0, [31:8]

RES0	Reserved.

DES_0, [7:4]

This is the least significant nibble of JEP106 ID code.

`0xB`	Arm Limited.

Part_1, [3:0]

Most significant nibble of the part number:

- For the cluster, the value is `0x4`.
- For the PEs, the value is a copy of bits [15:12] of the MIDR of the corresponding PE.

Configurations

The CTIPIDR1 is in the Debug power domain and is optional to implement in the external register interface.

Usage constraints

Accessing the CTIPIDR1

The CTIPIDR1 can be accessed through the external debug interface with offset `0xFE4`.

Accessibility

The accessibility of the CTIPIDR1 by condition code is:

Default
RO

See 16.11 External register access permissions on page 214 for the condition codes.

16.5 CTIPIDR2, CTI Peripheral Identification Register 2

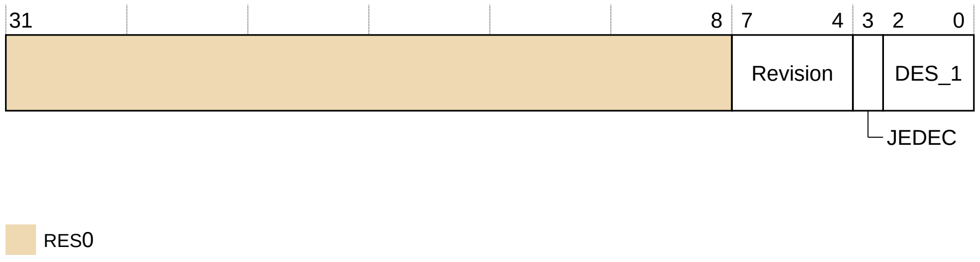
The CTIPIDR2 register provides information to identify a CTI component.

Bit field descriptions

CTIPIDR2 is a 32-bit register.

This register is Read Only.

Figure 16-3: CTIPIDR2 bit assignments



RES0, [31:8]

RES0	Reserved.

Revision, [7:4]

`0x7`	r4p1

JEDEC, [3]

`1`	**RES1**. Indicates that a JEP106 identity code is used.

DES_1, [2:0]

`0b011`	Arm Limited. This is the most significant nibble of JEP106 ID code.

Configurations

The CTIPIDR2 is in the Debug power domain and is optional to implement in the external register interface.

Usage constraints

Accessing the CTIPIDR2

The CTIPIDR2 can be accessed through the external debug interface with offset `0xFE8`.

Accessibility

The accessibility of the CTIPIDR2 by condition code is:

Default
RO

See 16.11 External register access permissions on page 214 for the condition codes.

16.6 CTIPIDR3, CTI Peripheral Identification Register 3

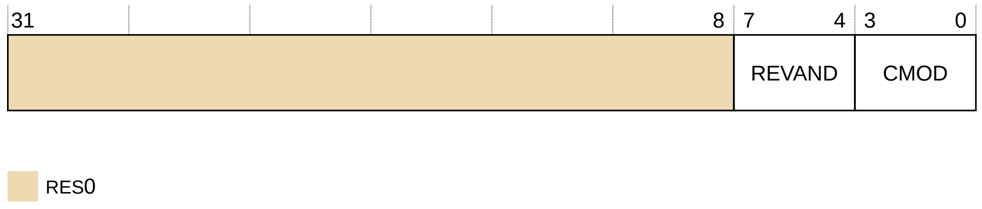
The CTIPIDR3 register provides information to identify a CTI component.

Bit field descriptions

CTIPIDR3 is a 32-bit register.

This register is Read Only.

Figure 16-4: CTIPIDR3 bit assignments



RES0, [31:8]

RES0	Reserved.

REVAND, [7:4]

0x0	Part minor revision.

CMOD, [3:0]

0x0	Customer modified.

Configurations

The CTIPIDR3 is in the Debug power domain and is optional to implement in the external register interface.

Usage constraints

Accessing the CTIPIDR3

The CTIPIDR3 can be accessed through the external debug interface with offset `0xFEC`.

Accessibility

The accessibility of the CTIPIDR3 by condition code is:

Default
RO

See 16.11 External register access permissions on page 214 for the condition codes.

16.7 CTIPIDR4, CTI Peripheral Identification Register 4

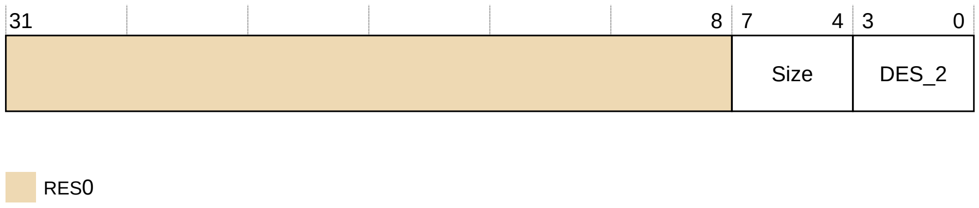
The CTIPIDR4 register provides information to identify a CTI component.

Bit field descriptions

CTIPIDR4 is a 32-bit register.

This register is Read Only.

Figure 16-5: CTIPIDR4 bit assignments



RES0, [31:8]

RES0	Reserved.

Size, [7:4]

0x0	Size of the component. Log_2 of 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 of the JEP106 continuation code.

Configurations

The CTIPIDR4 is in the Debug power domain and is optional to implement in the external register interface.

Usage constraints

Accessing the CTIPIDR4

The CTIPIDR4 can be accessed through the external debug interface with offset `0xFD0`.

Accessibility

The accessibility of the CTIPIDR4 by condition code is:

Default
RO

See 16.11 External register access permissions on page 214 for the condition codes.

16.8 CTIITCTRL, CTI Integration Mode Control Register

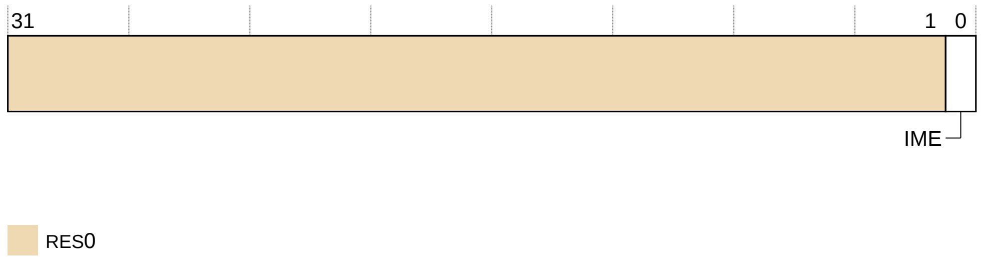
The CTIITCTRL register shows that the DSU does not implement an integration mode.

Bit field descriptions

CTIPIDR4 is a 32-bit register.

This register is Read Only.

Figure 16-6: CTIITCTRL bit assignments



RES0, [31:1]

RES0	Reserved.

IME, [0]

Integration mode enable. The value is:

0	Normal operation.

Configurations

The CTIITCTRL register is in the Debug power domain.

Usage constraints

Accessing the CTIITCTRL

The CTIITCTRL register can be accessed through external debug interface with offset `0xF00`.

Accessibility

The accessibility of the CTIITCTRL register by condition code is:

Default
RO

See 16.11 External register access permissions on page 214 for the condition codes.

16.9 CTIDEVAFF0, Cluster CTI Device Affinity register 0

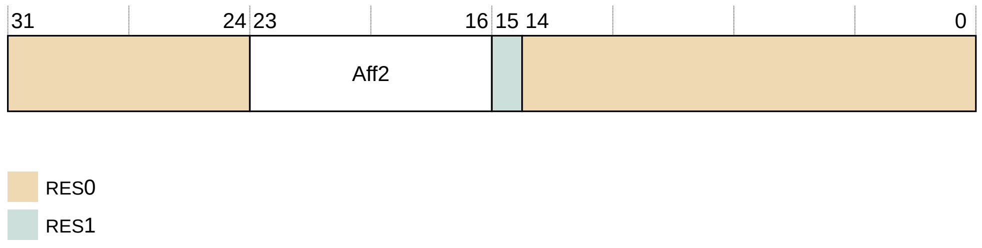
The CTIDEVAFF0 register identifies that the CTI relates to the DSU cluster.

Bit field descriptions

CTIDEVAFF0 is a 32-bit register.

This register is Read Only.

Figure 16-7: CTIDEVAFF0 bit assignments



RES0, [31:24]

RES0	Reserved.

Aff2, [23:16]

ClusterID Affinity Level-2 field.

The value is set by the CLUSTERIDAFF2[7:0] configuration input bus.

RES1, [15]

RES1	Reserved.

RES0, [14:0]

RES0	Reserved.

Configurations

The CTIDEVAFF0 is in the Debug power domain and is optional to implement in the external register interface.

Usage constraints

Accessing the CTIDEVAFF0

The CTIDEVAFF0 can be accessed through the external debug interface with offset `0xFA8`.

Accessibility

The accessibility of the CTIDEVAFF0 by condition code is:

Default
RO

See 16.11 External register access permissions on page 214 for the condition codes.

16.10 CTIDEVID, CTI Device Identification Register

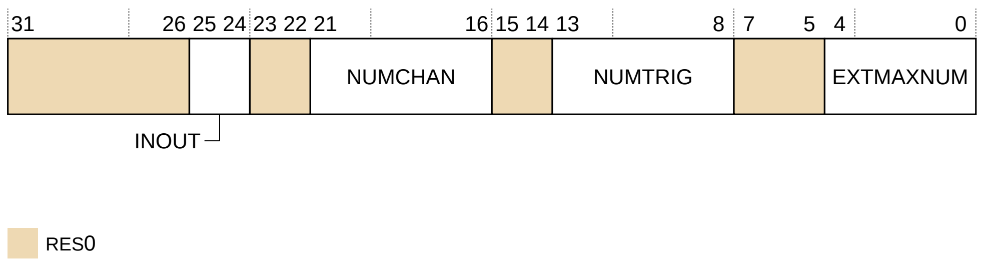
The CTIDEVID register describes the CTI component to the debugger.

Bit field descriptions

CTIDEVID is a 32-bit register.

This register is Read Only.

Figure 16-8: CTIDEVID bit assignments



RES0, [31:26]

RES0	Reserved.

INOUT, [25:24]

Input and output options. Indicates the presence of an input gate. This value is:

`0b01`	CTIGATE masks propagation of input events from external channels.

RES0, [23:22]

RES0	Reserved.

NUMCHAN, [21:16]

Number of channels implemented. This value is:

`0b000100`	Four channels implemented.

RES0, [15:14]

RES0	Reserved.

NUMTRIG, [13:8]

Number of triggers implemented. This value is:

`0b001010`	Ten triggers implemented.

RES0, [7:5]

RES0	Reserved.

EXTMAXNUM, [4:0]

Maximum number of external triggers implemented.This value is:

`0b00000`	No external triggers implemented.

Configurations

The CTIDEVID register is in the Debug power domain.

Usage constraints

Accessing the CTIDEVID

The CTIDEVID register can be accessed through the external debug interface with offset `0xFC8`.

Accessibility

The accessibility of the CTIDEVID register by condition code is:

Default
RO

See 16.11 External register access permissions on page 214 for the condition codes.

16.11 External register access permissions

External access permission to the CTI registers is subject to the conditions at the time of the access. The following table describes the response of the core to accesses through the external debug and memory-mapped interfaces.

Table 16-12: 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.
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 EDPRSR.SDAD 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 CTI 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 16-13: External register condition code example

Off	DLK	OSLK	EDAD	Default
-	-	-	-	RO

17. ROM table

This chapter describes the CoreSight™ ROM Table component.

17.1 About the ROM table

The ROM table holds the locations of debug components.

The ROM table complies with the *Arm® CoreSight™ Architecture Specification v3.0*. This table contains a list of components such as PE debug units, *Cross Trigger Interfaces* (CTIs), PE *Performance Monitoring Units* (PMUs), and *Embedded Trace Macrocells* (ETMs). Debuggers can use the ROM table to determine which components are implemented.

If a component is not included in your implementation, the corresponding ROM table entry indicates that the component is not present.

17.2 ROM table register summary

The specific implementation of your macrocell determines the contents of the ROM table.

The following table shows the offsets from the physical base address of the ROM table. The register formats are described in the *Arm® CoreSight™ Architecture Specification v3.0*.

Table 17-1: ROM table registers

Offset	Name	Reset value	Description
`0x000-0x144`	ROMENTRY0-81	-[12]	ROM entries
`0x148-0x9FC`	-	`0x00000000`	Reserved
`0xA00`	DBGPCR0	• `0x00000000` if Cluster ELA is not implemented. • `0x00000001` if Cluster ELA is implemented.	Debug Power Control Registers
`0xA04`	DBGPCR1	• `0x00000000` if Core0 ELA is not implemented. • `0x00000001` if Core0 ELA is implemented.	
`0xA08`	DBGPCR2	• `0x00000000` if Core1 ELA is not implemented. • `0x00000001` if Core1 ELA is implemented.	

Offset	Name	Reset value	Description
0xA0C	DBGPCR3	• 0x00000000 if Core2 ELA is not implemented. • 0x00000001 if Core2 ELA is implemented.	
0xA10	DBGPCR4	• 0x00000000 if Core3 ELA is not implemented. • 0x00000001 if Core3 ELA is implemented.	
0xA14	DBGPCR5	• 0x00000000 if Core4 ELA is not implemented. • 0x00000001 if Core4 ELA is implemented.	
0xA18	DBGPCR6	• 0x00000000 if Core5 ELA is not implemented. • 0x00000001 if Core5 ELA is implemented.	
0xA1C	DBGPCR7	• 0x00000000 if Core6 ELA is not implemented. • 0x00000001 if Core6 ELA is implemented.	
0xA20	DBGPCR8	• 0x00000000 if Core7 ELA is not implemented. • 0x00000001 if Core7 ELA is implemented.	
0xA24-0xA7C	-	0x00000000	Reserved
0xA80	DBGPSR0	0x00000000	Debug Power Status Registers
0xA84	DBGPSR1	0x00000000	
0xA88	DBGPSR2	0x00000000	
0xA8C	DBGPSR3	0x00000000	
0xA90	DBGPSR4	0x00000000	
0xA94	DBGPSR5	0x00000000	
0xA98	DBGPSR6	0x00000000	
0xA9C	DBGPSR7	0x00000000	
0xAA0	DBGPSR8	0x00000000	
0xAA4-0xBFC	-	0x00000000	Reserved
0xC00	PRIDR0	0x00000001	Power Reset Identification Register 0

Offset	Name	Reset value	Description
`0xC04-0xFB4`	-	`0x00000000`	Reserved
`0xFB8`	AUTHSTATUS	`0x00000008`	Authentication Status Register
`0xFBC`	DEVARCH	`0x47700AF7`	Device Architecture Register
`0xFC0-0xFC4`	-	`0x00000000`	Reserved
`0xFC8`	DEVID	`0x00000020`	Device ID Register
`0xFCC`	DEVTYPE	`0x00000000`	Device Type Register
`0xFD0`	PIDR4	`0x00000004`	Peripheral Identification Register 4
`0xFD4-0xFDC`	-	`0x00000000`	Reserved
`0xFE0`	PIDR0	`0x000000E3` [13]	Peripheral Identification Register 0
		`0x000000E4` [14]	
`0xFE4`	PIDR1	`0x000000B4`	Peripheral Identification Register 1
`0xFE8`	PIDR2	`0x0000007B`	Peripheral Identification Register 2
`0xFEC`	PIDR3	`0x00000000`	Peripheral Identification Register 3
`0xFF0`	CIDR0	`0x0000000D`	Component Identification Register 0
`0xFF4`	CIDR1	`0x00000090`	Component Identification Register 1
`0xFF8`	CIDR2	`0x00000005`	Component Identification Register 2
`0xFFC`	CIDR3	`0x000000B1`	Component Identification Register 3

[12] The ROMENTRY entry values depend on the number and type of cores implemented.
[13] If v7 debug memory map is implemented.
[14] If v8 debug memory map is implemented.

A Appendixes

This section contains the appendixes for the signal descriptions and compatible core information for the *DynamIQ™ Shared Unit* (DSU).

Appendix A Compatible Core Versions

This appendix provides the location of where to obtain information about the permissible combinations of cores.

A.1 Compatible Core Versions

For information on the number and type of permissible cores in the DSU, see the appendix `Compatible core versions` in the *Arm® DynamIQ™ Shared Unit Configuration and Sign-off Guide*.

Note

The *Arm® DynamIQ™ Shared Unit Configuration and Sign-off Guide* is a confidential document only available to licensees.

Appendix B Signal descriptions

This appendix describes the DSU signals.

B.1 Signal naming convention

Signals and buses are named using the following convention:

- Some signals or buses are per-core or per-thread. For single-threaded cores, there is a one-to-one mapping between cores and threads.
- Signals specified with a width of [CN:0] (<signal>[CN:0]) have 1 bit per core, where CN is the number of cores minus 1 (for example, CN =3 for a quad-core system).
- Signals named <signal>x have an instance of the signal per core, where x takes values from 0 to the number of cores minus 1.
- Signals specified with a width of [PE:0] (<signal>[PE:0]) have 1 bit per thread, where PE is the total number of threads minus one (for example PE=5 for a cluster with two single-threaded cores and two dual-threaded cores).
- Signals named <signal>y have an instance of the signal per thread, where y takes values from 0 to the total number of threads minus 1.

Signal widths

The DSU supports cores that might have different PA widths. The PA width determines how much physical memory the core can access and is a fixed value for each type of core.

There are signals with variable widths that depend on the PA size. In this case, the width is given either as [p:0] or [q:0]. If there are different cores in the cluster with different PA widths, then the PA width of the cluster matches the largest PA width of the configured cores.

The following table shows the PA widths that the DSU supports and the corresponding p and q values.

Table A-1: p and q values for signal widths

PA width	p	q
40	39	43
44	43	43
48	47	47
52	51	51

Note

The cluster configuration script outputs the maximum core PA width when the `--verbose` option is used.

In some cases, there are signals with variable widths that depend on the configuration. In this case, the width is given as [i:0], [j:0], [k:0], or [d:0]. The following tables show the respective values for various configurations.

Table A-2: Signal width i, j, k values for CHI.B, CHI.C, and CHI.D

CHI	i	j	k
CHI.B	58	85	48
CHI.C	58	85	48
CHI.D	65	100	63

Table A-3: CHI signal widths d values

Interface	Cache Protection	d for CHI.B	d for CHI.C	d for CHI.D
128-bit	N	212	213	-
	Y	214	215	-
256-bit	N	356	357	365
	Y	360	361	369

B.2 DSU signals

This section shows the *DynamIQ™ Shared Unit* (DSU) signals.

B.2.1 Clock and clock enable signals

This section describes the clock and clock enable signals.

Table A-4: Clock signals

Signal	Direction	Description
CORECLK[CN:0]	Input	Clock for each core. **Note:** If a core is synchronous to the cluster, then the corresponding bit of this signal is not present and the core uses SCLK instead. If all cores are synchronous to the cluster, then this signal is not present.
SCLK	Input	Clock for the SCU/L3 and the AMBA interface.
PCLK	Input	Clock for the debug APB interface.
ATCLK	Input	Clock for the ATB trace interface.
GICCLK	Input	Clock for the GIC interface.
PERIPHCLK	Input	Clock for the timers, power management, and other miscellaneous logic.

Table A-5: Clock enable signals

Signal	Direction	Description
ACLKENM	Input	ACE Master bus clock enable. This pin is only present when the DSU is configured with the ACE interface.

Signal	Direction	Description
ACLKENMP	Input	AXI Master peripheral port clock enable. This pin is only present when the DSU is configured with the peripheral port interface.
ACLKENS	Input	AXI Slave bus clock enable. This pin is only present when the DSU is configured with the ACP interface.
CNTCLKEN	Input	Counter clock enable.
SCLKENM	Input	CHI Master bus clock enable. This pin is only present when the DSU is configured with the CHI interface.
TSCLKEN	Input	Timestamp clock enable.

B.2.2 Reset signals

This section describes the reset signals.

Table A-6: Reset signals

Signal	Direction	Description
nATRESET	Input	Active-LOW reset for all registers in the ATCLK domain.
nCORERESET[CN:0]	Input	Active-LOW core Warm reset, excluding Debug, RAS, and ETM.
nCPUPORESET[CN:0]	Input	Active-LOW core Cold reset.
nGICRESET	Input	Active-LOW reset for all registers in the GICCLK domain.
nMBISTRESET	Input	Active-LOW reset for all MBIST logic.
nPERIPHRESET	Input	Active-LOW reset for all registers in the PERIPHCLK domain.
nPRESET	Input	Active-LOW reset for all registers in the PCLK domain.
nSPORESET	Input	Active-LOW Cold reset for all registers in the SCLK domain.
nSRESET	Input	Active-LOW Warm reset for all registers in the SCLK domain, excluding RAS registers.

B.2.3 Configuration signals

This section describes the configuration signals.

Table A-7: Configuration signals

Signal	Direction	Description
AA64nAA32[PE:0]	Input	Register width state. The options are: **0** AArch32 **1** AArch64 **Note:** This signal is only present for cores which support AArch32 execution at EL3. For example, if there are two single-threaded cores, and only core 1 supports executing AArch32 at EL3, then only bit[1] of this signal is present. If none of the cores support AArch32 execution at EL3, this signal is not present.
CFGEND[PE:0]	Input	Endianness configuration, controls the reset value of the SCTLR_EL3/SCTLR EE bit. The options are: **0** EE bit is LOW. **1** EE bit is HIGH.

Signal	Direction	Description
CFGTE[PE:0]	Input	Enable Thumb exceptions, controls the reset value of the SCTLR.TE bit. The options are: **0** TE is LOW. **1** TE is HIGH. **Note:** This signal is only present for cores which support AArch32 execution at EL3. For example, if there are two single-threaded cores, and only core 1 supports executing AArch32 at EL3, then only bit[1] of this signal is present. If none of the cores support AArch32 execution at EL3, this signal is not present.
CLUSTERIDAFF2[7:0]	Input	Value read in ClusterID Affinity Level-2 field, MPIDR bits[23:16].
CLUSTERIDAFF3[7:0]	Input	Value read in ClusterID Affinity Level-3 field, MPIDR bits[39:32].
CRYPTODISABLE	Input	Disables the Cryptographic Extensions. **Note:** If Cryptographic Extensions are not enabled, this signal is not present.
GICCDISABLE	Input	Globally disables the CPU interface logic and routes the external interrupt signals directly to the cores.
RVBARADDRy[p:2]	Input	Reset Vector Base Address for executing in 64-bit state.
VINITHI[PE:0]	Input	Enable high exception vectors, controls the reset value of the SCTLR.V bit. The options are: **0** Starts exceptions vectors at address `0x00000000`. **1** Starts exceptions vectors at address `0xFFFF0000`. **Note:** This signal is only present for cores which support AArch32 execution at EL3. For example, if there are two single-threaded cores, and only core 1 supports executing AArch32 at EL3, then only bit[1] of this signal is present. If none of the cores support AArch32 execution at EL3, this signal is not present.

B.2.4 GIC signals

This section describes the GIC and AXI4 Stream Protocol signals.

The following table shows the GIC signals.

Table A-8: GIC signals

Signal	Direction	Description
nFIQ[PE:0]	Input	Active-LOW, level-sensitive fast interrupt request.
nIRQ[PE:0]	Input	Active-LOW, level-sensitive interrupt request.
nVFIQ[PE:0]	Input	Active-LOW, level-sensitive virtual fast interrupt request.
nVIRQ[PE:0]	Input	Active-LOW, level-sensitive virtual interrupt request.
nVCPUMNTIRQ[PE:0]	Output	Active-LOW, level-sensitive virtual CPU interface maintenance interrupt PPI output.

The following table shows the AXI4 Stream Protocol signals.

Table A-9: AXI4 Stream Protocol signals

Signal	Direction	Description
IRITVALID	Input	Distributor to GIC CPU Interface messages. TVALID indicates that the master is driving a valid transfer.
IRITREADY	Output	Distributor to GIC CPU Interface messages. TREADY indicates that the slave can accept a transfer in the current cycle.
IRITDATA[15:0]	Input	Distributor to GIC CPU Interface messages. TDATA is the primary payload that is used to provide the data that is passing across the interface.
IRITLAST	Input	Distributor to GIC CPU Interface messages. TLAST indicates the boundary of a packet.
IRITDEST[3:0]	Input	Distributor to GIC CPU Interface messages. TDEST provides routing information for the data stream. Depending on the cluster configuration, not all bits of this bus are used. Any unused upper bits should be tied LOW.
IRITWAKEUP	Input	AXI4 stream protocol activity indicator.
ICCTVALID	Output	GIC CPU Interface to distributor messages. TVALID indicates that the master is driving a valid transfer.
ICCTREADY	Input	GIC CPU Interface to distributor messages. TREADY indicates that the slave can accept a transfer in the current cycle.
ICCTDATA[15:0]	Output	GIC CPU Interface to distributor messages. TDATA is the primary payload that is used to provide the data that is passing across the interface.
ICCTLAST	Output	GIC CPU Interface to distributor messages. TLAST indicates the boundary of a packet.
ICCTID[3:0]	Output	GIC CPU Interface to distributor. TID is the data stream identifier that indicates different streams of data.
ICCTWAKEUP	Output	AXI4 Stream Protocol activity indicator.

B.2.5 Generic Timer signals

This section describes the Generic Timer signals.

Table A-10: Generic Timer signals

Signal	Direction	Description
nCNTHPIRQ[PE:0]	Output	Active-LOW, level-sensitive Hypervisor physical timer event.
nCNTPNSIRQ[PE:0]	Output	Active-LOW, level-sensitive Non-secure physical timer event.
nCNTPSIRQ[PE:0]	Output	Active-LOW, level-sensitive Secure physical timer event.
nCNTVIRQ[PE:0]	Output	Active-LOW, level-sensitive virtual timer event.
nCNTHVIRQ[PE:0]	Output	Active-LOW, level-sensitive Hypervisor virtual timer event.
CNTVALUEB[63:0]	Input	Counter value in binary encoding.

B.2.6 Power management signals

This section describes the power management signals.

Note

The PPMCTLx[3:0] signal is only present for certain cores. See the associated core Technical Reference Manual for details.

Table A-11: Power management signals

Signal	Direction	Description
ATCLKQACCEPTn	Output	Active-LOW signal that indicates that the cluster accepts the clock controller request.
ATCLKQACTIVE	Output	Indicates that the cluster requires ATCLK to be active.
ATCLKQDENY	Output	Active-LOW signal that indicates that the cluster denies the clock controller request.
ATCLKQREQn	Input	Active-LOW signal that indicates that the clock controller wants to gate the clock.
CLUSTERPACCEPT	Output	Indicates that the cluster accepts the power controller request.
CLUSTERPACTIVE[19:0]	Output	Indicates if the cluster is active in various power states.
CLUSTERPDENY	Output	Indicates that the cluster denies the power controller request.
CLUSTERPREQ	Input	Indicates that the power controller wants the cluster to move to a new power state.
CLUSTERPSTATE[6:0]	Input	Power state that the power controller requires the cluster to move to.
COREPACCEPTx	Output	Indicates that the referenced core accepts the power controller request.
COREPACTIVEx[17:0]	Output	Indicates if the referenced core is active in various power states.
COREPDENYx	Output	Indicates that the referenced core denies the power controller request.
COREPREQx	Input	Indicates that the power controller wants the referenced core to move to a new power state.
COREPSTATEx[5:0]	Input	Power state that the power controller requires the core to move to.
EVENTIACK	Output	Event input request acknowledge. It is not asserted until EVENTIREQ is HIGH, and then remains asserted until after EVENTIREQ goes LOW.
EVENTIREQ	Input	Event input request for wake up from WFE state. It must remain asserted until EVENTIACK is asserted, and must not be re-asserted until EVENTIACK is LOW.
EVENTOACK	Input	Event output request acknowledge. It must not be asserted until EVENTOREQ is HIGH, and then must remain asserted until after EVENTOREQ goes LOW.
EVENTOREQ	Output	Event output request for wake up, triggered by SEV instruction. It is only asserted when EVENTOACK is LOW, and then remains HIGH until after EVENTOACK goes HIGH.
GICCLKQACCEPTn	Output	Active-LOW signal that indicates that the cluster accepts the clock controller request.
GICCLKQACTIVE	Output	Indicates that the cluster requires GICCLK to be active.
GICCLKQDENY	Output	Indicates that the cluster denies the clock controller request.
GICCLKQREQn	Input	Active-LOW signal that indicates that the clock controller wants the gate the clock.
PCLKQACCEPTn	Output	Active-LOW signal that indicates that the cluster accepts the clock controller request.
PCLKQACTIVE	Output	Indicates that the cluster requires PCLK to be active.
PCLKQDENY	Output	Indicates that the cluster denies the clock controller request.
PCLKQREQn	Input	Active-LOW signals that indicate that the clock controller wants the gate the clock.
SCLKQACCEPTn	Output	Active-LOW signal that indicates that the cluster accepts the clock controller request.
SCLKQACTIVE	Output	Indicates that the cluster requires SCLK to be active.
SCLKQDENY	Output	Indicates that the cluster denies the clock controller request.
SCLKQREQn	Input	Active-LOW signal that indicates that the clock controller wants the gate the clock.

Signal	Direction	Description
MPMMEN[PE:0]	Input	*Maximum Power Mitigation Mechanism* (MPMM) enable signal. The options are: **0** MPMM disabled. **1** MPMM enabled. See the associated core Technical Reference Manual for details of the MPMM behavior if supported on your core. **Note:** This signal is only present for cores that support MPMM. For example, if there are two single-threaded cores, and only core 1 supports MPMM, then only one bit of this signal is present. If none of the cores support MPMM, then this signal is not present.
MPMMSTATEx[1:0]	Input	MPMM state signal. If MPMM is enabled, selects which MPMM configuration to apply. If MPMM is supported on your core, see the associated core Technical Reference Manual for details of the MPMM configuration selection. **Note:** This signal is only present for cores that support MPMM configuration selection based on input pins. Not all cores that support MPMM support this.
PPMCTLx[3:0]	Input	Power policy control signal. **Note:** This signal is only present for certain cores. See the associated core Technical Reference Manual for details.

B.2.7 Error signals

This section describes the error signals.

Table A-12: Timing constraints for the error signals

Signal	Direction	Description
nFAULTIRQ[CN+1:1]	Output	Active-LOW, level-sensitive fault indicator for a detected 1-bit or 2-bit ECC or Parity error in the RAMs. Bits [CN+1:1] are for the L1 and L2 RAMs in each core. The bits are output from the PERIPHCLK domain.
nFAULTIRQ[0]	Output	Active-LOW, level-sensitive fault indicator for a detected 1-bit or 2-bit ECC or Parity error in the RAMs. Bit [0] is for L3 or snoop filter RAMs. It is output from the SCLK domain.
nERRIRQ[CN+1:1]	Output	Active-LOW, level-sensitive error indicator for an ECC error that causes potential data corruption or loss of coherency. Bits [CN+1:1] are for the L1 and L2 RAMs in each core. The bits are output from the PERIPHCLK domain.
nERRIRQ[0]	Output	Active-LOW, level-sensitive error indicator for an ECC error that causes potential data corruption or loss of coherency. Bit [0] is for L3 or snoop filter RAMs or ACE or CHI write transactions with a write response condition. It is output from the SCLK domain.

B.2.8 ACP interface signals

This interface exists only if the DSU is configured to have the ACP interface.

Table A-13: ACP wakeup signal

Signal	Direction	Description
AWAKEUPS	Input	ACP Slave activity indicator.

Table A-14: ACP write address channel signals

Signal	Direction	Description
AWREADYS	Output	Write address ready.
AWVALIDS	Input	Write address valid.
AWIDS[7:0]	Input	Write address ID.
AWADDRS[p:0]	Input	Write address.
AWLENS[7:0]	Input	Write burst length.
AWCACHES[3:0]	Input	Write cache type.
AWDOMAINS[1:0]	Input	Write attributes.
AWSNOOPS[3:0]	Input	Write request type.
AWPROTS[2:0]	Input	Write protection type.
AWSTASHLPIDS[3:0]	Input	Write stash target core.
AWSTASHLPIDENS	Input	Write stash target enable.

Table A-15: ACP write data channel signals

Signal	Direction	Description
WREADYS	Output	Write data ready.
WVALIDS	Input	Write data valid.
WDATAS[127:0]	Input	Write data.
WSTRBS[15:0]	Input	Write byte-lane strobes.
WLASTS	Input	Write data last transfer indication.

Table A-16: ACP write response channel signals

Signal	Direction	Description
BREADYS	Input	Write response ready.
BVALIDS	Output	Write response valid.
BIDS[7:0]	Output	Write response ID.
BRESPS[1:0]	Output	Write response.

Table A-17: ACP read address channel signals

Signal	Direction	Description
ARREADYS	Output	Read address ready.
ARVALIDS	Input	Read address valid.
ARIDS[7:0]	Input	Read address ID.

Signal	Direction	Description
ARADDRS[p:0]	Input	Read address.
ARLENS[7:0]	Input	Read burst length.
ARCACHES[3:0]	Input	Read cache type.
ARDOMAINS[1:0]	Input	Read attributes.
ARPROTS[2:0]	Input	Read protection type.

Table A-18: ACP read data channel signals

Signal	Direction	Description
RREADYS	Input	Read data ready.
RVALIDS	Output	Read data valid.
RIDS[7:0]	Output	Read data ID.
RDATAS[127:0]	Output	Read data.
RRESPS[1:0]	Output	Read data response.
RLASTS	Output	Read data last transfer indication.

B.2.9 Peripheral port interface signals

This interface exists only if the DSU is configured to have a peripheral port interface.

Table A-19: Peripheral port wakeup signals

Signal	Direction	Description
AWAKEUPMP	Output	AXI activity indicator.

Table A-20: Peripheral port configuration signals

Signal	Direction	Description
ASTARTMP[p:20]	Input	Start address for peripheral port address range.
AENDMP[p:20]	Input	End address for peripheral port address range.

Table A-21: Peripheral port write address channel signals

Signal	Direction	Description
AWREADYMP	Input	Write address ready.
AWVALIDMP	Output	Write address valid.
AWIDMP[3:0]	Output	Write address ID.
AWADDRMP[p:0]	Output	Write address.
AWCACHEMP[3:0]	Output	Write cache type.
AWDOMAINMP[1:0]	Output	Write protection type.
AWPROTMP[2:0]	Output	Write protection type.
AWSIZEMP[2:0]	Output	Write size.

Table A-22: Peripheral port write data channel signals

Signal	Direction	Description
WREADYMP	Input	Write data ready.
WVALIDMP	Output	Write data valid.
WDATAMP[63:0]	Output	Write data.
WSTRBMP[7:0]	Output	Write byte-lane strobes.
WLASTMP	Output	Write data last transfer indication.

Table A-23: Peripheral port write response channel signals

Signal	Direction	Description
BREADYMP	Output	Write response ready.
BVALIDMP	Input	Write response valid.
BIDMP[3:0]	Input	Write response ID.
BRESPMP[1:0]	Input	Write response.

Table A-24: Peripheral port read address channel signals

Signal	Direction	Description
ARREADYMP	Input	Read address ready.
ARVALIDMP	Output	Read address valid.
ARIDMP[3:0]	Output	Read address ID.
ARADDRMP[p:0]	Output	Read address.
ARCACHEMP[3:0]	Output	Read cache type.
ARPROTMP[2:0]	Output	Read protection type.
ARSIZEMP[2:0]	Output	Read size.
ARDOMAINMP[1:0]	Output	Write attributes.

Table A-25: Peripheral port read data channel signals

Signal	Direction	Description
RREADYMP	Output	Read data ready.
RVALIDMP	Input	Read data valid.
RIDMP[3:0]	Input	Read data ID.
RDATAMP[63:0]	Input	Read data.
RRESPMP[1:0]	Input	Read data response.

B.2.10 Broadcast signals for the memory interface

The signals in the following table are common to the memory interface, irrespective of whether the ACE or CHI protocol is used.

Table A-26: Broadcast signals

Signal	Direction	Description
BROADCASTCACHEMAINT	Input	Enable broadcasting of cache maintenance operations to downstream caches. The options are: **0** Cache maintenance operations are not broadcast to downstream caches. **1** Cache maintenance operations are broadcast to downstream caches.
BROADCASTCACHEMAINTPOU	Input	Enable broadcasting of cache maintenance operations to the point of unification. The options are: **0** Cache maintenance operations DCCMVAU and `DC CVAU` are not broadcast to other clusters. This is more efficient if all other clusters are Arm Cortex processors. **1** Cache maintenance operations DCCMVAU and `DC CVAU` are broadcast to other clusters.
BROADCASTPERSIST	Input	Enable broadcasting of cache clean to the point of persistence operations. The options are: **0** `DC CVAP` instructions are treated the same as `DC CVAC`. **1** `DC CVAP` instructions are sent as a clean to the point of persistence transaction externally.
BROADCASTOUTER	Input	Enable broadcasting of Outer Shareable transactions. The options are: **0** Outer Shareable transactions are not broadcast externally. **1** Outer Shareable transactions are broadcast externally. **Note:** Within the cluster, Inner Shareable and Outer Shareable memory are treated identically. Therefore, this signal also controls the broadcast of Inner Shareable transactions.

B.2.11 ACE interface signals

This interface exists only if the DSU is configured to have the ACE master interface.

The following tables show the signals present when one ACE interface is implemented. The signals have a suffix of M0. If a second ACE interface is implemented, then there is an additional set of ACE signals, suffixed by M1.

Table A-27: ACE write address channel signals

Signal	Direction	Description
AWADDRM0[q:0] [15]	Output	Write address.
AWBURSTM0[1:0]	Output	Write burst type.

[15] The value of q is 43 when p is 39. For any other value of p, q is the same as p.

Signal	Direction	Description
AWCACHEM0[3:0]	Output	Write cache type.
AWDOMAINM0[1:0]	Output	Write shareability domain type.
AWIDM0[7:0]	Output	Write address ID.
AWLENM0[7:0]	Output	Write burst length.
AWLOCKM0	Output	Write lock type.
AWPROTM0[2:0]	Output	Write protection type.
AWREADYM0	Input	Write address ready.
AWSIZEM0[2:0]	Output	Write burst size.
AWSNOOPM0[2:0]	Output	Write snoop request type.
AWUNIQUEM0	Output	For WriteBack, WriteClean and WriteBackUC transactions. Indicates that the write is Unique (`0b1`) or shared (`0b0`).
AWVALIDM0	Output	Write address valid.

Table A-28: ACE write data channel signals

Signal	Direction	Description
WDATAM0[127:0]	Output	Write data.
WIDM0[7:0]	Output	Write data ID. Only used when connecting to an AXI3 interconnect.
WLASTM0	Output	Write data last transfer indication.
WREADYM0	Input	Write data ready.
WSTRBM0[15:0]	Output	Write byte-lane strobes.
WVALIDM0	Output	Write data valid.

Table A-29: ACE write data response channel signals

Signal	Direction	Description
BIDM0[7:0]	Input	Write response ID.
BREADYM0	Output	Write response valid.
BRESPM0[1:0]	Input	Write response.
BVALIDM0	Input	Write response valid.

Table A-30: ACE read address channel signals

Signal	Direction	Description
ARADDRM0[q:0] [15]	Output	Read address.
ARBURSTM0[1:0]	Output	Read burst type.
ARCACHEM0[3:0]	Output	Read cache type.
ARDOMAINM0[1:0]	Output	Read shareability domain type.
ARIDM0[8:0]	Output	Read address ID.
ARLENM0[7:0]	Output	Read burst length.
ARLOCKM0	Output	Read lock type.
ARPROTM0[2:0]	Output	Read protection type.
ARREADYM0	Input	Read address ready.
ARSIZEM0[2:0]	Output	Read burst size.

Signal	Direction	Description
ARSNOOPM0[3:0]	Output	Read snoop request type.
ARVALIDM0	Output	Read address valid.
ARVMIDEXTM0[3:0]	Output	Additional VMID bits for DVM messages.

Table A-31: ACE read data channel signals

Signal	Direction	Description
RDATAM0[127:0]	Input	Read data.
RIDM0[8:0]	Input	Read data ID.
RLASTM0	Input	Read data last transfer indication.
RREADYM0	Output	Read data ready.
RRESPM0[3:0]	Input	Read data response.
RVALIDM0	Input	Read data valid.

Table A-32: ACE coherency address channel signals

Signal	Direction	Description
ACADDRM0[q:0] [15]	Input	Snoop address.
ACPROTM0[2:0]	Input	Snoop protection type.
ACREADYM0	Output	Master ready to accept snoop address.
ACVMIDEXTM0[3:0]	Input	Additional VMID bits for DVM messages.
ACSNOOPM0[3:0]	Input	Snoop request type.
ACVALIDM0	Input	Snoop address valid.

Table A-33: ACE coherency response channel signals

Signal	Direction	Description
CRREADYM0	Input	Slave ready to accept snoop response.
CRVALIDM0	Output	Snoop response valid.
CRRESPM0[4:0]	Output	Snoop response.

Table A-34: ACE coherency data channel handshake signals

Signal	Direction	Description
CDDATAM0[127:0]	Output	Snoop data.
CDLASTM0	Output	Snoop data last transfer.
CDREADYM0	Input	Slave ready to accept snoop data.
CDVALIDM0	Output	Snoop data valid.

Table A-35: ACE read and write acknowledge signals

Signal	Direction	Description
RACKM0	Output	Read acknowledge.
WACKM0	Output	Write acknowledge.

Table A-36: ACE low-power wakeup signals

Signal	Direction	Description
AWAKEUPM0	Output	ACE pending activity indicator.
ACWAKEUPM0	Input	ACE snoop activity indicator.

Table A-37: ACE coherency signals

Signal	Direction	Description
SYSCOREQM0	Output	The cluster requests to be part of the coherency domain and can safely receive snoops.
SYSCOACKM0	Input	The system might send snoops to the cluster.

Table A-38: ACE SRCATTRM0/1 bus signals

Signal	Direction	Description
RDSRCATTRM0[11:0]	Output	These signals provide additional information about the transactions. The functionality of the individual bits is described in the following table.
WRSRCATTRM0[11:0]	Output	
RDSRCATTRM1[11:0]	Output	
WRSRCATTRM1[11:0]	Output	

Table A-39: ACE SRCATTRM0/1 bus signal functionality

Bits	Field	Description	Encoding
[11:7]	Source	Source of request. This field has the same encoding as TXREQ[LPID] field on CHI.	The possible values are: **`0x0-0xF`** Core ID/Thread ID Bit 0 is the thread ID, or 0 for single threaded processors. **`0x1E`** ACP **`0x1F`** L3 cache eviction
[6]	Outer cacheable	Outer cacheability of access.	**0** Outer non-cacheable (or device). **1** Outer cacheable (WT/WB)

Bits	Field	Description	Encoding
[5]	Stream_or_transient	The exact conditions for this bit depend on the core, which indicates that this access was either: • Generated by a non-temporal instruction, or • Generated as part of a detected stream, or • Marked as transient in the translation tables. **Note:** This bit only applies to accesses that are directly part of the stream, and not to cache evictions that are a side effect of the stream.	`0` Normal `1` Streaming or transient
[4]	Prefetch	Demand or Prefetch.	`0` Demand `1` Prefetch
[3:2]	Subsource	Instruction, or Data or Table walk.	The possible values are: `0b00` Instruction `0b01` Data This includes all miscellaneous transaction types, for example CMOs and DVMs. `0b10` Table walk
[1:0]	PBHA	Page-based hardware attributes.	

B.2.12 CHI interface signals

This interface exists only if the DSU is configured to have the CHI master interface.

Common signals for single or dual CHI interfaces

The following signal is present in both single or dual CHI configurations.

Note

For information on the signal width naming convention values, see B.1 Signal naming convention on page 221.

Table A-40: CHI common configuration signal

Signal	Direction	Description
BROADCASTATOMIC[16].	Input	Enable broadcasting of atomic transactions. The options are: **0** Atomic transactions are not sent externally. **1** Atomic transactions are sent externally.

Per-interface signals for single or dual CHI interfaces

The following tables show the signals present for either a single CHI interface, or the signals for CHI interface 0 in a dual CHI implementation. If a second CHI interface is implemented, then there is an additional set of CHI signals, suffixed by M1. For example, in a dual CHI configuration, the signals for the DSU cluster CHI Node Identifiers for the two CHI interfaces are NODEID[10:0] and NODEIDM1[10:0].

Table A-41: CHI configuration signals

Signal	Direction	Description
NODEID[10:0]	Input	DSU cluster CHI Node Identifier.

Table A-42: CHI activity signals

Signal	Direction	Description
RXSACTIVE	Input	Receive pending activity indicator.
TXSACTIVE	Output	Transmit pending activity indicator.
RXLINKACTIVEREQ	Input	Receive link active request.
RXLINKACTIVEACK	Output	Receive link active acknowledge.
TXLINKACTIVEREQ	Output	Transmit link active request.
TXLINKACTIVEACK	Input	Transmit link active acknowledge.

Table A-43: CHI transmit request virtual channel signals

Signal	Direction	Description
TXREQFLITPEND	Output	Transmit Request Flit pending.
TXREQFLITV	Output	Transmit Request Flit valid.
TXREQFLIT[j+q:0]	Output	Transmit Request Flit payload.
TXREQLCRDV	Input	Transmit Request link-layer credit valid.

Table A-44: CHI transmit response virtual channel signals

Signal	Direction	Description
TXRSPFLITPEND	Output	Transmit response flit pending.
TXRSPFLITV	Output	Transmit response flit valid.
TXRSPFLIT[i:0]	Output	Transmit response flit.
TXRSPLCRDV	Input	Transmit response link-layer credit valid.

[16] This signal is sampled on reset like the other broadcast signals that are listed in B.2.10 Broadcast signals for the memory interface on page 230

Table A-45: CHI transmit data virtual channel signals

Signal	Direction	Description
TXDATFLITPEND	Output	Transmit Data Flit pending.
TXDATFLITV	Output	Transmit Data Flit valid.
TXDATFLIT[d:0]	Output	Transmit Data Flit.
TXDATLCRDV	Input	Transmit Data link-layer credit valid.

Table A-46: CHI receive snoop virtual channel signals

Signal	Direction	Description
RXSNPFLITPEND	Input	Receive Snoop Flit pending.
RXSNPFLITV	Input	Receive Snoop Flit valid.
RXSNPFLIT[k+q:0]	Input	Receive Snoop Flit.
RXSNPLCRDV	Output	Receive Snoop link-layer credit valid.

Table A-47: CHI receive response virtual channel signals

Signal	Direction	Description
RXRSPFLITPEND	Input	Receive Response Flit pending.
RXRSPFLITV	Input	Receive Response Flit valid.
RXRSPFLIT[i:0]	Input	Receive Response Flit.
RXRSPLCRDV	Output	Receive Response link-layer credit valid.

Table A-48: CHI receive data virtual channel signals

Signal	Direction	Description
RXDATFLITPEND	Input	Receive Data Flit pending.
RXDATFLITV	Input	Receive Data Flit valid.
RXDATFLIT[d:0]	Input	Receive Data Flit.
RXDATLCRDV	Output	Receive Data link-layer credit valid.

Table A-49: CHI coherency signals

Signal	Direction	Description
SYSCOREQ	Output	The cluster requests to be part of the coherency domain and can safely receive snoops.
SYSCOACK	Input	The system might send snoops to the cluster.

Table A-50: CHI REQSRCATTR bus signals

Signal	Direction	Description
REQSRCATTR[6:0]	Output	These signals provide additional information about the transactions. The functionality of the individual bits is described in the following table.

Table A-51: CHI REQSRCATTR bus signal functionality

Bits	Field	Description	Encoding
[6]	Outer cacheable	Outer cacheability of access.	`0` Outer non-cacheable (or device). `1` Outer cacheable (WT/ WB)
[5]	Stream_or_transient	The exact conditions for this bit depend on the core, which indicates that this access was either: • Generated by a non-temporal instruction, or • Generated as part of a detected stream, or • Marked as transient in the translation tables. **Note:** This bit only applies to accesses that are directly part of the stream, and not to cache evictions that are a side effect of the stream.	`0` Normal `1` Streaming or transient
[4]	Prefetch	Demand or Prefetch.	`0` Demand `1` Prefetch
[3:2]	Subsource	Instruction, or Data or Table walk.	The possible values are: `0b00` Instruction `0b01` Data This includes all miscellaneous transaction types, for example, CMOs and DVMs. `0b10` Table walk
[1:0]	PBHA	Page-based hardware attributes.	

B.2.12.1 System address map signals

This interface exists only if the DSU is configured to have the CHI interface and the CMN-600 SAM option is enabled. In a dual CHI configuration, there is only one set of system address map signals.

Table A-52: CHI system address map signals

Signal	Direction	Description
RXPUBCFGACTIVE	Input	SAM configuration pending activity indicator.
RXPUBFLITPEND	Input	Receive Data Flit pending.

Signal	Direction	Description
RXPUBFLITV	Input	Receive Data Flit valid.
RXPUBFLIT[34:0]	Input	Receive Data Flit.
RXPUBLCRDV_RP1	Output	Receive Data link-layer credit valid.
RXPUBLINKACTIVEREQ	Input	Receive link active request.
RXPUBLINKACTIVEACK	Output	Receive link active acknowledge.
RXPUBLINKFLIT	Input	Receive link-layer flit.
TXPUBFLITPEND	Output	Transmit Data Flit pending.
TXPUBFLITV	Output	Transmit Data Flit valid.
TXPUBFLIT[34:0]	Output	Transmit Data Flit.
TXPUBLCRDV_RP1	Input	Transmit Data link-layer credit valid.
TXPUBLINKACTIVEREQ	Output	Transmit link active request.
TXPUBLINKACTIVEACK	Input	Transmit link active acknowledge.
TXPUBLINKFLIT	Output	Transmit link-layer flit.

Table A-53: CHI system address map configuration signals

Signal	Direction	Description
RNSAM_CFGM_DESTID[15:0]	Input	CMN-600 CFG master destination ID.
RNSAM_CFGS_NODEID[15:0]	Input	Source ID for the cluster CFG slave.
RNSAM_DEFAULT_TGTID[10:0]	Input	Default target ID of the boot node.
RNSAM_DN_TGTID[10:0]	Input	Target ID for the DN that handles DVMs.
RNSAM_PGMED_RDY	Input	Reset value of the Programmed Ready bit.

B.2.13 DebugBlock APB interface signals

The interface between the DebugBlock and the cluster consists of a pair of APB interfaces, one in each direction. The signals on the cluster interface should be connected to the equivalently named signal on the DebugBlock.

The following table shows the signals between the cluster APB master and the DebugBlock APB slave.

Table A-54: Cluster to DebugBlock APB interface signals

Signal	Direction	Description
PSELCD	Output	APB CD select.
PADDRCD[8:2]	Output	APB address.
PENABLECD	Output	APB enable.
PWDATACD[8:0]	Output	APB write data.
PREADYCD	Input	APB slave ready, used to extend a transfer.
PSLVERRCD	Input	APB slave transfer error.
PWAKEUPCD	Output	APB activity indicator.

The following table shows the signals between the DebugBlock APB master and the cluster APB slave.

Table A-55: DebugBlock to cluster APB interface signals

Signal	Direction	Description
PSELDC	Input	APB DC select.
PADDRDC[19:2]	Input	APB address.
PENABLEDC	Input	APB enable.
PWRITEDC	Input	APB read/write indicator. When asserted, indicates a write.
PWDATADC[31:0]	Input	APB write data.
PRDATADC[31:0]	Output	APB read data.
PREADYDC	Output	APB slave ready, used to extend a transfer.
PSLVERRDC	Output	APB slave transfer error.
PWAKEUPDC	Input	APB activity indicator.

The following table shows the miscellaneous debug signals.

Table A-56: Miscellaneous debug signals

Signal	Direction	Description
DBGRSTREQ[PE:0]	Output	Debug reset request.
nCOMMIRQ[PE:0]	Output	Active-LOW, level-sensitive comms channel receive or transmit interrupt request.
DBGEN	Input	Invasive debug enable.
NIDEN	Input	Non-invasive debug enable.
SPIDEN	Input	Secure privilege invasive debug enable.
SPNIDEN	Input	Secure privilege non-invasive debug enable.
DBGCONNECTED	Input	A debugger is connected and so the DebugBlock can be accessed on boot.

B.2.14 ATB interface signals

The following table shows the ATB interface signals.

Table A-57: ATB interface signals

Signal	Direction	Description
ATREADYMx	Input	ATB device ready.
AFVALIDMx	Input	FIFO flush request.
ATDATAMx[31:0]	Output	Data.
ATVALIDMx	Output	Data valid.
ATBYTESMx[1:0]	Output	Data size.
AFREADYMx	Output	FIFO flush finished
ATIDMx[6:0]	Output	Trace source ID.
ATWAKEUPMx	Output	ATB activity indicator.
SYNCREQMx	Input	Synchronization request.

B.2.15 Timestamp signal

This section describes the timestamp signal.

The TSVALUEB input provides the incrementing count value that is required for generating the trace timestamp for each core. The DSU includes an interpolator on the TSVALUEB input for each of the cores.

Table A-58: Timestamp signal

Signal	Direction	Description
TSVALUEB[63:0]	Input	Timestamp in binary encoding.

B.2.16 PMU interface signals

This section describes the PMU interface signals.

Table A-59: PMU interface signals

Signal	Direction	Description
COREINSTRRET[PE:0]	Output	Indicates that a core has retired at least one instruction recently. This signal is output from the PERIPHCLK domain.
COREINSTRRUN[PE:0]	Output	Indicates that a core is in a running state (powered up and not in WFI or WFE low-power states). This signal is output from the PERIPHCLK domain.
nCLUSTERPMUIRQ	Output	Active-LOW, level-sensitive Cluster PMU interrupt request. This signal is output from the SCLK domain.
nPMBIRQ[PE:0]	Output	Active-LOW, level-sensitive SPE interrupt request. This signal only exists for cores that support the architectural *Statistical Profiling Extension* (SPE). This signal is output from the PERIPHCLK domain.
nPMUIRQ[PE:0]	Output	Active-LOW, level-sensitive PMU interrupt request.
PMUSNAPSHOTACK	Output	Acknowledge a snapshot request. This signal forms a four-phase handshake with PMUSNAPSHOTREQ.
PMUSNAPSHOTREQ	Input	Request for a snapshot of the PMU counters. When asserted, this signal must remain HIGH until PMUSNAPSHOTACK is asserted. It must not be reasserted until PMUSNAPSHOTACK is deasserted.

B.2.17 ELA signal

This signal is present only if ELA-500 support is configured.

Table A-60: ELA signal

Signal	Direction	Description
STOPCLOCK	Output	ELA request for the system to stop the clocks. This signal is generated from multiple clock domains and must be synchronized in the system before use.

B.2.18 DFT interface signals

This section describes the DFT signals.

Table A-61: DFT interface signals

Signal	Direction	Description
DFTRAMHOLD	Input	Disables the RAM chip select during scan shift.
DFTRSTDISABLE[1:0]	Input	Disables internal synchronized reset during scan shift.
DFTCGEN	Input	Forces on the clock gates during scan shift.
DFTMCPHOLD	Input	Disables Multicycle Paths on RAM interfaces.
DFTCORECLKDISABLE[CN:0]	Input	Disables specific CORECLK signals from toggling during scan, to reduce test power if only part of the design is being scanned.
DFTSCLKBYPASS	Input	Bypasses clock stretching logic on L3 data RAMs. **Note:** This signal is only present when the `L3_DATA_STRETCH_CLK` configuration option is enabled. See 2.3 Implementation options on page 18.

B.2.19 MBIST interface signals

This section describes the MBIST interface signals.

Table A-62: MBIST interface signals

Signal	Direction	Description
MBISTREQ	Input	MBIST test request.

B.3 DebugBlock signals

This section shows the DebugBlock signals.

B.3.1 Clock signal

The following table shows the clock signal.

Table A-63: Clock signal

Signal	Direction	Description
PCLK	Input	Clock for the DebugBlock and APB interface.

B.3.2 Reset signal

The following table shows the reset signal.

Table A-64: Reset signal

Signal	Direction	Description
nPRESET	Input	Reset for all registers in the DebugBlock.

B.3.3 Power and clock gate control signals

The following table shows the power and clock gate control signals.

Table A-65: Power and clock gate signals

Signal	Direction	Description
PDBGCLKQREQn	Input	Active-LOW signal that indicates that the clock controller wants to gate the clock.
PWRQREQn	Input	Active-LOW signal that indicates that the power controller wants to power down the DebugBlock.
PDBGCLKQACTIVE	Output	Indicates that the DebugBlock requires PCLK to be active.
PDBGCLKQACCEPTn	Output	Active-LOW signal that indicates that the DebugBlock accepts the clock controller request.
PDBGCLKQDENY	Output	Indicates that the DebugBlock denies the clock controller request.
PWRQACTIVE	Output	Indicates that the DebugBlock requires power.
PWRQACCEPTn	Output	Active-LOW signal that indicates that the DebugBlock accepts the power controller request. It causes all future APB accesses to the DebugBlock to receive a PSLVERR response.
PWRQDENY	Output	Indicates that the DebugBlock denies the power controller request because of an ongoing transaction.

B.3.4 Configuration signals

The following table shows the configuration signals. All these configuration pins are sampled only on reset.

Note

The following signals must be tied to the same values as the corresponding cluster signals.

Table A-66: Configuration signals

Signal	Direction	Description
CLUSTERIDAFF2[7:0]	Input	Value read in ClusterID Affinity Level-2 field, MPIDR bits [23:16].
CLUSTERIDAFF3[7:0]	Input	Value read in ClusterID Affinity Level-3 field, MPIDR bits [39:32].

Signal	Direction	Description
ELADISABLE	Input	Disables the ELA logic in the cluster. This signal is not present if ELA support is not implemented. **Note:** The ELA is intended only for silicon debug of sample devices. The ELADISABLE pin must be driven HIGH in any production devices.
GICCDISABLE	Input	Globally disables the CPU interface logic and routes the external signals directly to the cores.

B.3.5 Debug signals

This section describes the debug interface signals.

The following table shows the external debug APB interface signals.

Note

The PPROTDBG[2:0] signal is only included when the DSU is used with a core that implements the Arm®v8.4-A debug architecture.

Table A-67: External debug APB interface signals

Signal	Direction	Description
PSELDBG	Input	APB select.
PADDRDBG[23:2]	Input	APB address.
PADDRDBG31	Input	APB address bit[31]. Controls the ETM OS Lock mechanism. **Note:** Software lock is not supported.
PENABLEDBG	Input	APB enable.
PPROTDBG[2:0]	Input	APB protection type. Only present when a core is used that implements the Arm®v8.4-A debug architecture.
PWRITEDBG	Input	APB read/write indicator. When asserted, indicates a write.
PWDATADBG[31:0]	Input	APB write data.
PWAKEUPDBG	Input	APB activity indicator.
PRDATADBG[31:0]	Output	APB read data.
PREADYDBG	Output	APB slave ready, used to extend a transfer.
PSLVERRDBG	Output	APB slave transfer error.

The following table shows the debug authentication signals.

Note

The signals NIDEN and SPNIDEN are not present if the core implements the Arm®v8.4-A Debug architecture.

Table A-68: Debug authentication signals

Signal	Direction	Description
DBGEN	Input	Invasive debug enable.
NIDEN	Input	Non-invasive debug enable. Not present if core implements Arm®v8.4-A debug architecture.
SPIDEN	Input	Secure privilege invasive debug enable.
SPNIDEN	Input	Secure privilege non-invasive debug enable. Not present if core implements Arm®v8.4-A Debug.
DBGPWRUPREQ[PE:0]	Output	Request to power up a PE for debug.
CLUSTERDBGPWRUPREQ	Output	Request power to the cluster domain. Supports the powerup of Cluster-level ELA from the external debugger.

The following table shows the signals between the cluster APB master and the DebugBlock APB slave.

Table A-69: Cluster to DebugBlock APB signals

Signal	Direction	Description
PADDRCD[8:2]	Input	APB address.
PENABLECD	Input	APB enable.
PREADYCD	Output	APB slave ready, used to extend a transfer.
PSELCD	Input	APB select.
PSLVERRCD	Output	APB slave transfer error.
PWAKEUPCD	Input	APB activity indicator.
PWDATACD[8:0]	Input	APB write data.

The following table shows the signals between the DebugBlock APB master and the cluster APB slave.

Note

The PPROTDC[1:1] signal is only included when the DSU is used with a core that implements the Arm®v8.4-A debug architecture.

Table A-70: DebugBlock to cluster APB signals

Signal	Direction	Description
PADDRDC[19:2]	Output	APB address.
PENABLEDC	Output	APB enable.
PPROTDC[1:1]	Output	APB security state. Only present when a core is used that implements the Arm®v8.4-A debug architecture.
PRDATADC[31:0]	Input	APB read data.
PREADYDC	Input	APB slave ready, used to extend a transfer.

Signal	Direction	Description
PSELDC	Output	APB select.
PSLVERRDC	Input	APB slave transfer error.
PWAKEUPDC	Output	APB activity indicator.
PWDATADC[31:0]	Output	APB write data.
PWRITEDC	Output	APB read/write indicator. When asserted, indicates a write.

B.3.6 CTI interface signals

This section describes the CTI interface signals.

Table A-71: CTI interface signals

Signal	Direction	Description
CTICHIN[3:0]	Input	Channel In.
CTICHOUTACK[3:0]	Input	Channel Out acknowledge.
CTICHOUT[3:0]	Output	Channel Out.
CTICHINACK[3:0]	Output	Channel In acknowledge.
CISBYPASS	Input	Channel interface sync bypass.
CIHSBYPASS[3:0]	Input	Channel interface H/S bypass.
CTIIRQ[PE:0]	Output	Active-HIGH, edge-sensitive CTI interrupt.
CTIIRQACK[PE:0]	Input	CTI interrupt acknowledge.

B.3.7 DFT signals

The following table shows the DFT signals.

Table A-72: DFT interface signals

Signal	Direction	Description
DFTCGEN	Input	Forces on the clock gates during scan shift.
DFTRSTDISABLE[1:0]	Input	Disables internal synchronized reset during scan shift.

Appendix C Revisions

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

C.1 Revisions

This section describes the technical changes between released issues of this document.

Table A-73: Issue 0000-00

Change	Location
First release	-

Table A-74: Differences between issue 0000-00 and issue 0001-00

Change	Location
Added support for ACE and CHI.	The whole document

Table A-75: Differences between issue 0001-00 and issue 0002-00

Change	Location
Change of terminology, *Flexible Cluster Microarchitecture* (FCM) to *DynamIQ™ Shared Unit* (DSU)	The whole document
Updated CLUSTERPSTATE value	5.5.4 Power mode encoding on page 47
Added description of error protection of snoop filter tag RAMs	6.5 L3 cache ECC and parity on page 57
Added traps and enables descriptions to AARCH64 registers	11. Control registers on page 95
Updated register description	12.3 ERR1CTLR, Error Record Control Register on page 134
Updated revision field	16.5 CTIPIDR2, CTI Peripheral Identification Register 2 on page 208
Updated ROM table Peripheral Identification Register 2 address	17.2 ROM table register summary on page 216

Table A-76: Differences between issue 0001-00 and issue 0100-00

Change	Location
Added `L3_CACHE` implementation option	Configuration parameters table
Added `L3_DATA_STRETCH_CLK` implementation option	Configuration parameters table
Added `INTERLEAVE_ADDR_BIT` implementation option	Configuration parameters table
Added `ELA_RAM_ADDR_SIZE` implementation option	Configuration parameters table
Added new L3 cache operating capacity level	Section L3 cache partial powerdown
Added new section	L3 cache partitioning
Updated L3 cache stashing information	L3 cache stashing
Updated ACE interface information	Dual ACE interfaces
Updated ACP interface information	About the ACP
Changes to Cluster Configuration Register	CLUSTERCFR, Cluster Configuration Register

Change	Location
Changes to allocation of CTI input trigger events	CTI triggers
Changes to PMU events	PMU events
Changes to ROM table reset values	ROM table register summary
Changes to DFT interface signals	DFT interface signals
Changes to Cluster to DebugBlock APB signals	Cluster to DebugBlock APB signals

Table A-77: Differences between issue 0100-00 and 0200-00

Change	Location
Support added for three sets of cores of different microarchitecture.	2.1 About the DSU on page 16
ARMv8.2-A architecture cores supported.	2.2 Features on page 17
`NUM_LITTLE_CORES` parameter includes 5 cores.	2.3 Implementation options on page 18
`NUM_OTHER_CORES` parameter added.	2.3 Implementation options on page 18
`OTHER_CORE_TYPE` parameter added.	2.3 Implementation options on page 18
`L3_CACHE_SIZE` parameter extended to include 1.5MB cache size option.	2.3 Implementation options on page 18
`NUM_SLICES` parameter added.	2.3 Implementation options on page 18
Added note to clarify the use of `L3_DATA_STRETCH_CLK` parameter on performance.	2.3 Implementation options on page 18
Added note to clarify the use of `ASYNC_BRIDGE` parameter.	2.3 Implementation options on page 18
Added note to `ELA` parameter about the 2 methods of creating unique instances.	2.3 Implementation options on page 18
New section Cluster Configurations added.	2.3.1 Cluster configurations on page 22
AMBA 5 Issue C support added.	2.4 Supported standards and specifications on page 22
Summarized r2p0 main technical changes.	2.7 Product revisions on page 24
Added note about CPU bridge RTL.	3.1 Components on page 25
L3 cache 1.5MB cache size.	3.1 Components on page 25
Changes between power mode transitions MEM_RET and FUNC_RET.	5. Power management on page 37
Updated note on L3 cache slices.	6.7 Cache slices and portions on page 62
Updated description for Read ID Attribute in table Encodings for ARIDM[8:0].	7.4 ACE master interface attributes on page 68
Updated Snoop acceptance capability description in ACE channel properties table.	7.5 ACE channel properties on page 71
Added section CHI version.	8.2 CHI version on page 79
Updated Snoop acceptance capability description.	8.6 CHI channel properties on page 82.
Updated Cluster Main RevisionID Register (CLUSTERIDR) description for r2p0.	11.9 CLUSTERIDR, Cluster Main Revision ID Register on page 109
Added Cluster Count Enable Set Register (CLUSTERPMCNTENSET) description.	13.5 CLUSTERPMCNTENSET, Cluster Count Enable Set Register on page 154
Added Cluster Count Enable Clear Register (CLUSTERPMCNTENCLR) description.	13.6 CLUSTERPMCNTENCLR, Cluster Count Enable Clear Register on page 156
Added Cluster Overflow Flag Status Set Register (CLUSTERPMOVSSET) description.	13.7 CLUSTERPMOVSSET, Cluster Overflow Flag Status Set Register on page 158

Change	Location
Added Cluster Overflow Flag Status Clear Register (CLUSTERPMOVSCLR) description.	13.8 CLUSTERPMOVSCLR, Cluster Overflow Flag Status Clear Register on page 160
Added Cluster Event Counter Selection Register (CLUSTERPMSELR) description.	13.9 CLUSTERPMSELR, Cluster Event Counter Selection Register on page 162
Added Cluster Interrupt Enable Set Register (CLUSTERPMINTENSET) description.	13.10 CLUSTERPMINTENSET, Cluster Interrupt Enable Set Register on page 164
Added Cluster Interrupt Enable Clear Register (CLUSTERPMINTENCLR) description.	13.11 CLUSTERPMINTENCLR, Cluster Interrupt Enable Clear Register on page 166
Added Cluster Performance Monitors Cycle Counter (CLUSTERPMCCNTR) description.	13.12 CLUSTERPMCCNTR, Cluster Performance Monitors Cycle Counter on page 168
Added Cluster Selected Event Type Register (CLUSTERPMXEVTYPER) description.	13.13 CLUSTERPMXEVTYPER, Cluster Selected Event Type Register on page 169
Added Cluster Selected Event Counter Register (CLUSTERPMXEVCNTR) description.	13.14 CLUSTERPMXEVCNTR, Cluster Selected Event Counter Register on page 171
Updated Cluster Monitor Debug Configuration Register (CLUSTERPMMDCR) description.	13.15 CLUSTERPMMDCR, Cluster Monitor Debug Configuration Register on page 173
Updated Cluster Common Event Identification Register 0 (CLUSTERPMCEID0) description.	13.16 CLUSTERPMCEID0, Cluster Common Event Identification Register 0 on page 175
Updated Cluster Common Event Identification Register 1 (CLUSTERPMCEID1) description.	13.17 CLUSTERPMCEID1, Cluster Common Event Identification Register 1 on page 177
Added Cluster Claim Tag Set Register (CLUSTERCLAIMSET) description.	13.18 CLUSTERCLAIMSET, Cluster Claim Tag Set Register on page 179
Added Cluster Claim Tag Clear Register (CLUSTERCLAIMCLR) description.	13.19 CLUSTERCLAIMCLR, Cluster Claim Tag Clear Register on page 181
Updated Cluster Event Type Register (CLUSTERPMEVTYPER<n>) description.	13.20 CLUSTERPMEVTYPER<n>, Cluster Event Type Register on page 183
Added Cluster Event Counter Register (CLUSTERPMEVCNTR<n>) description.	13.21 CLUSTERPMEVCNTR<n>, Cluster Event Counter Register on page 184
Added signal MPMMEN	B.2.6 Power management signals on page 225

Table A-78: Differences between issue 0200-00 and issue 0300-00

Change	Location
Updated Implementation options table.	2.3 Implementation options on page 18
Updated Product revisions.	2.7 Product revisions on page 24
Updated L3 cache description to qualify which cache sizes are 12-way set associative and which are 16-way set associative.	3.1 Components on page 25
Updated DSU interfaces table to include two CHI interfaces.	3.2 Interfaces on page 28
Qualified resets as either Cold reset, or Warm reset.	Throughout document
Updated description of Off power mode.	5.1 About DSU power management on page 37
Updates description of Power mode control.	5.2 Power mode control on page 37
Updated description of L3 cache partial powerdown.	5.4.1 L3 cache partial powerdown on page 39
Updated the description of SFONLY FUNC_RET, ¼FUNC_RET, ½FUNC_RET, ¾FUNC_RET, FULL FUNC_RET.	5.5.1 Power mode transitions on page 41
Added note about the SFONLY , ¼ , ½ , ¾ and FULL DEBUG_RECOV modes.	5.5.1 Power mode transitions on page 41
Added Victim RAM and LTDB RAMs.	5.5.4 Power mode encoding on page 47

Change	Location
Added note about CLUSTERPACTIVE and COREPACTIVEx outputs.	5.6 Power operating requirements on page 48
Added section about power control for DFT.	5.6.1 Power control for DFT on page 49
Described the signal naming convention for dual CHI.	5.9.1.1 Coherency signals naming convention on page 53
Qualified how AWSTASHLPIDS signal indicates core and thread.	6.4 Cache stashing on page 56
Updated description of L3 cache data RAM latency.	6.6 L3 cache data RAM latency on page 60
Updated L3 cache slices section to include cache portions description.	6.7 Cache slices and portions on page 62
Added combined issuing capability to ACE master interface attributes.	7.4 ACE master interface attributes on page 68
Updated note.	7.9 Write response on page 76
Added dual CHI interface description.	8.1.1 Dual CHI interfaces on page 78
Updated CHI channel properties to account for dual CHI.	8.6 CHI channel properties on page 82
Updated CLUSTERCFR register description.	11.7 CLUSTERCFR, Cluster Configuration Register on page 101
Updated CLUSTERECTLR register description.	11.8 CLUSTERECTLR, Cluster Extended Control Register on page 105
Updated CLUSTERIDR register description.	11.9 CLUSTERIDR, Cluster Main Revision ID Register on page 109
Updated CLUSTERPWRCTLR register description.	11.13 CLUSTERPWRCTLR, Cluster Power Control Register on page 119
Updated PMU events table description to align with architecture description.	15.3 PMU events - DSU R3 and R3 ADP on page 193
Updated CTIPIDR2 register description.	16.5 CTIPIDR2, CTI Peripheral Identification Register 2 on page 208
Added MPMMSTATEx[1:0] signal description.	B.2.6 Power management signals on page 225
Updated CHI interface signals for the dual CHI interface.	B.2.12 CHI interface signals on page 235

Table A-79: Differences between issue 0300-00 and issue 0300-01

Change	Location
Qualified the use of L3_DATA_WR_LATENCY value of 2p.	2.3 Implementation options on page 18
Clarified how CTIs are allocated.	3.1 Components on page 25
Clarified the Cluster to DebugBlock interface.	3.2 Interfaces on page 28
Clarified the Direct connect connection.	3.5 L3 memory system variants on page 32
Updated ACE transactions.	7.6 ACE transactions on page 73
Updated the description for the WAY, [31:28] bit field in the ERR1MISC0 register description.	12.5 ERR1MISC0, Error Record Miscellaneous Register 0 on page 138
Updated DebugBlock diagram and description to clarify the Cluster CTI.	14.4 DebugBlock components on page 187, 14.5 About the Embedded Cross Trigger on page 188, 14.5.1 Supported debug and trace trigger events on page 190
Added Compatible Core Versions Appendix.	A. Compatible Core Versions on page 220

Table A-80: Differences between issue 0400-00 and issue 0300-01

Change	Location
Fixed various typographical errors.	The whole document.
Updated Features section to include Arm®v8.4-A support.	2.2 Features on page 17.

Change	Location
Updated Implementation options section to include new L3 cache sizes of 256KB and 3MB, and to document new `L3_PBHA` parameter.	2.3 Implementation options on page 18
Added Arm®v8.4-A core and debug support. Added GIC v4.0 support.	2.4 Supported standards and specifications on page 22
Updated product revisions for r4p0 to include support for CHI.D, 256KB L3 cache size, and PBHA.	2.7 Product revisions on page 24
Updated Components to include new cache sizes.	3.1 Components on page 25
Added Page Based Hardware Attribute description.	3.4 Page-based hardware attributes on page 31
Updated description of Direct connect.	3.5 L3 memory system variants on page 32
Updated CLUSTERPSTATE value	5.5.4 Power mode encoding on page 47
Updated memory retention description to explain Snoop Filter RAMs cannot be placed into retention.	5.4.2 L3 RAM retention on page 40
Updated section for 256KB and 3MB cache size.	6.7.3 Implementing a 1.5MB or 3MB L3 cache on page 64
Added new note to qualify issuing capability for the cluster.	7.4 ACE master interface attributes on page 68 and 8.6 CHI channel properties on page 82
Added new section describing support for memory types.	7.7 Support for memory types on page 75
Updated description to explain the additional signals to tie LOW for AXI compatibility mode.	7.11 AXI compatibility mode on page 76
Added new section.	8.8 Use of DataSource on page 86
Added new section.	7.7 Support for memory types on page 75
Added note about WriteUniquePtlStash.	9.4 ACP transaction types on page 89
Updated section.	9.5 ACP performance on page 91
Updated Variant bit-field in CLUSTERIDR register description.	11.9 CLUSTERIDR, Cluster Main Revision ID Register on page 109
Updated Revision bit-field in CTIPIDR2 register description.	16.5 CTIPIDR2, CTI Peripheral Identification Register 2 on page 208
Added new signals PPROTDBG[2:0], PPROTDC[1:1]	B.3.5 Debug signals on page 244
Added note about NIDEN and SPNIDEN signals	B.3.5 Debug signals on page 244

Table A-81: Differences between issue 0400-00 and issue 0400-01

Change	Location
There are no technical changes.	-

Table A-82: Differences between issue 0400-01 and 0400-02

Change	Location
Fixed various typographical errors.	The whole document.
Changed ACE-Lite to ACE5-Lite.	3.2 Interfaces on page 28
Corrected two misspelt PBHA acronyms.	3.4 Page-based hardware attributes on page 31

Change	Location
Corrected the first 4 rows of the 'L3 write bandwidth' column of the 'L3 data RAM performance' table.	6.6 L3 cache data RAM latency on page 60
Changed ACE to ACE5.	7.1 About the ACE master interface on page 66
Amended the 'CHI transaction usage' table.	8.7 CHI transactions on page 83
Changed ACE-Lite to ACE5-Lite.	9.1 About the ACP on page 88
Changed ACE-Lite to ACE5-Lite.	9.3 ACP ACE5-Lite subset on page 89
Better explained the Write-Allocate bit.	9.5 ACP performance on page 91
Added read/write access examples for both the AArch64 and AArch32 states to the 'Usage constraints' section of these Control registers.	11.4 CLUSTERACPSID, Cluster ACP Scheme ID Register on page 97, 11.6 CLUSTERBUSQOS, Cluster Bus QoS Control Register on page 99, 11.7 CLUSTERCFR, Cluster Configuration Register on page 101, 11.8 CLUSTERECTLR, Cluster Extended Control Register on page 105, 11.9 CLUSTERIDR, Cluster Main Revision ID Register on page 109, 11.10 CLUSTERL3HIT, Cluster L3 Hit Counter Register on page 111, 11.11 CLUSTERL3MISS, Cluster L3 Miss Counter Register on page 113, 11.12 CLUSTERPARTCR, Cluster Partition Control Register on page 114, 11.13 CLUSTERPWRCTLR, Cluster Power Control Register on page 119, 11.14 CLUSTERPWRDN, Cluster Powerdown Register on page 121, 11.15 CLUSTERPWRSTAT, Cluster Power Status Register on page 123, 11.16 CLUSTERREVIDR, Cluster Revision ID Register on page 125, 11.17 CLUSTERSTASHSID, Cluster Stash Scheme ID Register on page 127, 11.18 CLUSTERTHREADSID, Cluster Thread Scheme ID Register on page 129, and 11.19 CLUSTERTHREADSIDOVR, Cluster Thread Scheme ID Override Register on page 130
Updated the `WAY,[31:28]` description to include the LTDB RAM.	12.5 ERR1MISC0, Error Record Miscellaneous Register 0 on page 138
Added read/write access examples for both the AArch64 and AArch32 states to the 'Usage constraints' section of these PMU registers.	13.4 CLUSTERPMCR, Cluster Performance Monitors Control Register on page 151, 13.5 CLUSTERPMCNTENSET, Cluster Count Enable Set Register on page 154, 13.6 CLUSTERPMCNTENCLR, Cluster Count Enable Clear Register on page 156, 13.7 CLUSTERPMOVSSET, Cluster Overflow Flag Status Set Register on page 158, 13.8 CLUSTERPMOVSCLR, Cluster Overflow Flag Status Clear Register on page 160, 13.9 CLUSTERPMSELR, Cluster Event Counter Selection Register on page 162, 13.10 CLUSTERPMINTENSET, Cluster Interrupt Enable Set Register on page 164, 13.11 CLUSTERPMINTENCLR, Cluster Interrupt Enable Clear Register on page 166, 13.12 CLUSTERPMCCNTR, Cluster Performance Monitors Cycle Counter on page 168, 13.13 CLUSTERPMXEVTYPER, Cluster Selected Event Type Register on page 169, 13.14 CLUSTERPMXEVCNTR, Cluster Selected Event Counter Register on page 171, 13.15 CLUSTERPMMDCR, Cluster Monitor Debug Configuration Register on page 173, 13.16 CLUSTERPMCEID0, Cluster Common Event Identification Register 0 on page 175, 13.17 CLUSTERPMCEID1, Cluster Common Event Identification Register 1 on page 177, 13.18 CLUSTERCLAIMSET, Cluster Claim Tag Set Register on page 179, and 13.19 CLUSTERCLAIMCLR, Cluster Claim Tag Clear Register on page 181

Change	Location
Amended the 'ACE SRCATTRM0/1 bus signal functionality' table.	B.2.11 ACE interface signals on page 231
Amended the 'CHI REQSRCATTR bus signal functionality' table.	B.2.12 CHI interface signals on page 235

Table A-83: Differences between issue 0400-02 and 0401-03

Change	Location
Summarized the r4p1 main technical changes.	2.7 Product revisions on page 24
Updated the revised revision-related values.	11.9 CLUSTERIDR, Cluster Main Revision ID Register on page 109, 16.5 CTIPIDR2, CTI Peripheral Identification Register 2 on page 208, and 17.2 ROM table register summary on page 216
Updated the ACE and CHI signals due to the Direct connect changes.	B.2.11 ACE interface signals on page 231 and B.2.12 CHI interface signals on page 235
Updated the descriptions of the PMU events, from `0x0500` to `0x0504`, to specify the L3 prefetch.	15.3 PMU events - DSU R3 and R3 ADP on page 193
Removed all references to the L3 data cache and replaced them the more accurate designation of L3 unified cache.	5.7 Wait For Interrupt and Wait For Event on page 50 and 13.17 CLUSTERPMCEID1, Cluster Common Event Identification Register 1 on page 177
Added more specific connection information for interrupt signals.	B.2.4 GIC signals on page 224, B.2.5 Generic Timer signals on page 225, B.2.7 Error signals on page 227, B.2.13 DebugBlock APB interface signals on page 239, B.2.16 PMU interface signals on page 241, and B.3.6 CTI interface signals on page 246
Correctly specified the PPMCTLx signal.	B.2.6 Power management signals on page 225
Describe how an interpolator is included on the TSVALUEB input signal.	B.2.15 Timestamp signal on page 241
Amended the L3 cache section.	6.1 About the L3 cache on page 54, 6.3 L3 cache partitioning on page 54, 6.4 Cache stashing on page 56, 6.5 L3 cache ECC and parity on page 57, 6.6 L3 cache data RAM latency on page 60, 6.7 Cache slices and portions on page 62, 6.7.1 Cache slice and master port selection on page 63, 6.7.2 Default number of cache slices on page 63, and 6.7.3 Implementing a 1.5MB or 3MB L3 cache on page 64
Ensured the power modes use a consistent use of the same case.	5.5.1 Power mode transitions on page 41
Added the `POP_RAM` configuration parameter, which was missing from the table.	2.3 Implementation options on page 18

Change	Location
Amended the DebugBlock power states description to describe a workaround due to an erratum.	14.3 About the DebugBlock on page 186

Table A-84: Differences between Issue 0401-03 and Issue 0401-04

Change	Location
Added a note to clarify what the `OTHER_CORE_TYPE` is.	2.3 Implementation options on page 18
Updated master interface to main memory. The DSU AE can be configured with either 1 or 2 ACE, or 1 or 2 CHI interfaces.	3.2 Interfaces on page 28
Updated the nMBISTRESET signal description.	4.2 Resets on page 35
Updated L3 power domain CLUSTERPACTIVE 0 bit description.	5.6 Power operating requirements on page 48
Added text about stashing to an L3 when no L3 present.	6.4 Cache stashing on page 56
Added new section about injectable errors.	6.5 L3 cache ECC and parity on page 57
Reworked paragraph about peripheral port address range.	10.1 About the peripheral port on page 93
Updated CLUSTERTHREADSIDOVR, Cluster Thread Scheme ID Override Register is banked for each thread of execution.	11.19 CLUSTERTHREADSIDOVR, Cluster Thread Scheme ID Override Register on page 130