Arm® Architecture Reference Manual Supplement

Memory System Resource Partitioning and Monitoring (MPAM), for Armv8-A

arm

ARM DDI 0598A.b (ID070519)

Arm Architecture Reference Manual Supplement
Memory System Resource Partitioning and Monitoring (MPAM), for Armv8-A

Release Information

The following releases of this document have been made.

Release history

Date	Issue	Confidentiality	Change
30 October, 2018	A.a	Non-Confidential	EAC release
05 July, 2019	A.b	Non-Confidential	Updated EAC release

Proprietary Notice

Arm Limited. Company 02557590 registered in England.
110 Fulbourn Road, Cambridge, England CB1 9NJ.

LES-PRE-20349

In this document, where the term Arm is used to refer to the company it means "Arm or any of its subsidiaries as appropriate".

Confidentiality Status

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

Product Status

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

Web Address

http://www.arm.com

Contents

Preface

This preface introduces the MPAM Extension Architecture Specification v1.0. It contains the following sections:

About this book

This book is the *Architecture Specification* for the *MPAM Extension Architecture Specification v1.0*.

It specifies System registers and behaviors for generation of MPAM information in processing elements (PEs). It also specifies memory-mapped registers and standard types of resource control interfaces for Memory-System Components (MSCs). It also covers resource usage monitors for measuring usage by software.

Together, these facilities permit software both to observe memory-system usage and to allocate resources to software by running that software in a memory-system partition.

This document defines the MPAM Extension version 1.0. This MPAM extension is an optional extension to Armv8.2 and later versions. This document covers only the AArch64 Execution state. However, see *MPAM AArch32 interoperability* on page 6-66 for interoperation with AArch32.

This document primarily describes hardware architecture. As such, it does not usually include information on either the software needed to control these facilities or the ways to implement effective controls of the memory system using the parameters defined by this architecture. This document gives no guidance as to which of the optional features to implement at which of the MSCs.

Intended audience

This document targets the following audience:

- Hardware and software developers interested in the MPAM hardware architecture.

Using this book

This book is organized into the following chapters:

Readers new to MPAM should first read Chapters 1 to 5.

Readers interested in MPAM generation behavior in the PE should read Chapters 6 and 7.

Readers interested in MPAM resource controls and memory-system component behaviors should read Chapters 8, 9, 11, 12, and Appendices A and B.

Readers interested in MPAM resource usage monitoring should read Chapters 10, 11, and 12.

Readers interested in pseudocode definition, refer to the "Arm® Architecture Reference Manual".

Chapter 1 *Introduction*

Read this chapter for an introduction to the MPAM extension.

Chapter 2 *MPAM and Arm Memory-System Architecture*

Read this chapter for a description of MPAM and Arm Memory-System Architecture.

Chapter 3 *ID Types, Properties, and Spaces*

Read this chapter for a description of ID Types, Properties, and Spaces.

Chapter 4 *MSC Propagation of MPAM Information*

Read this chapter for a description of MSC Propagation of MPAM Information.

Chapter 5 *System Model*

Read this chapter for a description of the System model.

Chapter 6 *PE Generation of MPAM Information*

Read this chapter for a description of PE Generation of MPAM Information.

Chapter 7 *System Registers*

Read this chapter for a description of the System registers.

Chapter 8 *MPAM in MSCs*

Read this chapter for a description of MPAM in MSCs.

Chapter 9 *Resource Partitioning Controls*

Read this chapter for a description of Memory-System Partitioning.

Chapter 10 *Resource Monitors*

Read this chapter for a description of Performance Monitoring Groups.

Chapter 11 *Memory-Mapped Registers*

Read this chapter for a description of Memory-Mapped Registers.

Chapter 12 *Errors in MSCs*

Read this chapter for a description of Errors in MSCs.

Chapter 13 *Armv8 Pseudocode*

Read this chapter for the pseudocode definitions that describe various features of the MPAM Architecture.

Appendix A *Generic Resource Controls*

Read this appendix for a description of Generic Resource Controls.

Appendix B *MSC Firmware Data*

Read this appendix for a description of MSC Firmware Data.

Glossary

Read this glossary for definitions of some of the terms that are used in this manual. The Arm Glossary does not contain terms that are industry standard unless the Arm meaning differs from the generally accepted meaning.

Note

Arm publishes a single glossary that relates to most Arm products, see the *Arm Glossary* http://infocenter.arm.com/help/topic/com.arm.doc.aeg0014-/index.html. A definition in the glossary in this book might be more detailed than the corresponding definition in the *Arm Glossary*.

Conventions

The following sections describe conventions that this book can use:

- *Typographic conventions*.
- *Signals*.
- *Numbers*.
- *Pseudocode descriptions*.

Typographic conventions

The typographical conventions are:

italic Introduces special terminology, and denotes citations.

bold Denotes signal names, and is used for terms in descriptive lists, where appropriate.

`monospace` Used for assembler syntax descriptions, pseudocode, and source code examples.

Also used in the main text for instruction mnemonics and for references to other items appearing in assembler syntax descriptions, pseudocode, and source code examples.

SMALL CAPITALS

Used for a few terms that have specific technical meanings, and are included in the Glossary LINK.

Colored text Indicates a link. This can be:

- A URL, for example `http://infocenter.arm.com`.
- A cross-reference, that includes the page number of the referenced information if it is not on the current page, for example, *Pseudocode descriptions*.
- A link to a chapter or appendix, or to a glossary entry, or to the section of the document that defines the colored term.

Signals

In general this specification does not define processor signals, but it does include some signal examples and recommendations.

The signal conventions are:

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

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

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

Numbers

Numbers are normally written in decimal. Binary numbers are preceded by `0b`, and hexadecimal numbers by `0x`. In both cases, the prefix and the associated value are written in a `monospace` font, for example `0xFFFF0000`.

Pseudocode descriptions

This book uses a form of pseudocode to provide precise descriptions of the specified functionality. This pseudocode is written in a `monospace` font, and is described in Appendix C *Pseudocode Definition*.

Additional reading

This section lists relevant publications from Arm and third parties.

See the Infocenter http://infocenter.arm.com, for access to Arm documentation.

Arm publications

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

- *Arm Architecture Reference Manual Armv8, for Armv8-A architecture profile* (ARM DDI 0487)
- *Armv8.1 Extensions* (ARM DDI0557)
- *Armv8.2 Extensions*
- *Armv8.3 Extensions* (ARM-ECM-0454992)
- *Armv8.4 Extensions* (ARM-ECM-0629567)
- *Arm CoreSight Architecture Specification v2.0* (ARM IHI029D)

Other publications

The following books are referred to in this book, or provide more information:

- *"Heracles: Improving Resource Efficiency at Scale,"* David Lo, Liqun Cheng, Rama Govindaraju, Parthasarathy Ranganathan, Christos Kozyrakis, 42nd Annual International Symposium on Computer Architecture (ISCA), New York NY, ACM, 2015.

Feedback

Arm welcomes feedback on its documentation.

Feedback on this book

If you have comments on the content of this book, send an e-mail to `errata@arm.com`. Give:

- The title.
- The number, ARM DDI 0598A.b.
- The page numbers to which your comments apply.
- A concise explanation of your comments.

Arm also welcomes general suggestions for additions and improvements.

Note

Arm tests PDFs only in Adobe Acrobat and Acrobat Reader, and cannot guarantee the appearance or behavior of any document when viewed with any other PDF reader.

Chapter 1
Introduction

This chapter contains the following sections:

1.1 Overview

Some shared-memory computer systems run multiple applications or multiple virtual machines (VMs) concurrently. Such systems may have one or more of the following needs:

- Control the performance effects of misbehaving software on the performance of other software.
- Bound the performance impact on some software by any other software.
- Minimize the performance impact of some software on other software.

These scenarios are common in enterprise networking and server systems. The Memory System Resource Partitioning and Monitoring (MPAM) extension addresses these scenarios with two approaches that work together, under software control, to apportion the performance-giving resources of the memory system. The apportionment can be used to align the division of memory-system performance between software, to match higher-level goals for dividing the performance of the system between software environments.

These approaches are:

- Memory-system resource partitioning.
- Memory-system resource usage monitoring.

The main motivation of the extension is to make data centers less expensive. The extension can increase server utilization, so that fewer servers are needed for a given level of service. Utilization can be increased by controlling how much impact the best-effort jobs have on the tail latency of responses by web-facing jobs. See Heracles: Improving Resource Efficiency at Scale.

This MPAM Extension describes:

- A mechanism for attaching partition identifiers and a monitoring property, for executing software on an Arm processing element (PE).
- Propagation of a Partition ID (PARTID) and Performance Monitoring Group (PMG) through the memory system.
- A framework for memory-system component controls that partition one or more of the performance resources of the component.
- Extension of the framework for MSCs to have performance monitoring that is sensitive to a combination of PARTID and PMG.
- Some implementation-independent, memory-mapped interfaces to memory-system component controls for performance resource controls most likely to be deployed in systems.
- Some implementation-independent memory-mapped interfaces to memory-system component resource monitoring that would likely be needed to monitor the partitioning of memory-system resources.

1.2 Memory-system resource partitioning

The performance of programs running on a computer system is affected by the memory-system performance, which is in part controlled by several resources in the memory system. In a memory system shared by multiple VMs, OSs, and applications, the resources available to one software environment may vary, depending on which other programs are also running. This is true because those other programs may consume more or less of an uncontrolled memory-system resource.

Memory-system resource partitioning provides controls on the limits and use of previously uncontrolled memory-system resources.

Shared, partitionable memory-system resources that can affect performance of a VM, OS, or application include:

- Shared caches, in which one application may displace the cached data of another application.
- Interconnect bandwidth, in which use by one application can interfere with use by another application due to contention for buffers, communication links, or other interconnect resources.
- Memory bandwidth, in which use by one application can interfere with the use by another application due to contention for DRAM bus bandwidth.

This list is not exhaustive. MPAM functionality can be extended through vendor and implementation-specific resource partitioning controls or resource-usage monitors.

Memory-system performance resource partitioning is performed by MPAM resource controls located within the MSCs. Each memory-system component may implement zero or more MPAM resource controls within that component.

An MPAM resource control uses the PARTID that is set for one or more software environments. A PARTID for the current software environment labels each memory system request. Each MPAM resource control has control settings for each PARTID. The PARTID in a request selects the control settings for that PARTID, which are then used to control the partitioning of the performance resources of that memory-system component.

1.3 Memory-system resource usage monitoring

Memory-system resource-usage monitoring measures memory-system resource usage. MSCs can have resource monitors. An MPAM monitor must be configured and enabled before it can be queried for resource-usage information. A monitor can be configured to be sensitive to a particular PARTID, or PARTID and PMG, and some monitors can be configured to certain subcategories of the resource (for example, the memory bandwidth used by writes that use a PARTID and PMG).

A monitor can measure resource usage or capacity usage, depending on the resource. For example, a cache can have monitors for cache storage that measure the usage of the cache by a PARTID and PMG.

Monitors can serve several purposes. A memory-system resource monitor might be used to find software environments to partition. Or, a monitor's reads might be used to tune the memory-system partitioning controls. A PMG value can be used to subdivide the software environments within a PARTID for finer-grained results, or to make measurements over prospective partitions.

1.4 Memory-system components

A Memory-System Component (MSC) is a function, unit, or design block in a memory system that can have partitionable resources. MSCs consist of all units that handle load or store requests issued by any MPAM master. These include cache memories, interconnects, Memory Management Units, memory channel controllers, queues, buffers, rate adaptors, and so on.

An MSC may be a part of another system component. For example, a PE may contain caches, which may contain MSCs.

1.5 Implementation flexibility

Memory-system partitioning, monitoring capabilities, and certain implementation parameters must be discoverable by software, and they must be used by software to adapt to the system hardware. Discovery of MPAM memory-system component topology is expected to be by means of firmware data such as Device Tree or ACPI interface. MPAM controls and parameters of MSCs are discoverable in memory-mapped ID registers. Discovery of PE MPAM features and parameters is described in *Discovering MPAM* on page 6-55.

The width of memory-system partitioning and monitoring values communicated through the system can be sized to the needs of the system. The costs can thereby be adjusted to meet the market requirements.

This document defines standard interfaces to some resource partitioning and monitoring features of MSCs. It does so by defining ID registers that expose implementation parameters and options. It also defines configuration registers that allow standard programming of these features while giving substantial implementation flexibility. In addition, this document also defines a mechanism that permits IMPLEMENTATION DEFINED partitioning and monitoring features that may introduce partitioning or monitoring in new ways or of new resource types.

1.6 Example uses

This section is *informative*. It presents examples of partitioning uses that reduce memory-system interactions.

1.6.1 Separate systems combined

With faster processors, it is often less expensive to integrate into a single computer system the functions previously performed by two or more systems. If any of these previously separate systems was real-time or otherwise performance-sensitive, it may be necessary to isolate the performance of that function from others in the integrated system.

Memory system performance can be monitored, and the measured usage can guide optimization of system partitioning.

Partitioning is often statically determined by the system developer. Partitions may be given non-shared resource allocations to improve real-time predictability. The number of partitions required could be small, similar to the number of previously separate systems.

1.6.2 Foreground and background job optimization

When foreground and background jobs are run on the same system, the foreground job's response time should not be compromised, and the background job's throughput should be optimized. The performance of the foreground and background jobs can be monitored, and the resource allocations can be changed dynamically to track system loading while optimizing foreground response time and background throughput.

An example of this approach is proposed in *Heracles: Improving Resource Efficiency at Scale*. This paper describes a system that requires only two partitions, one for web-facing applications and another for best-effort applications. The Heracles approach measures the service-level objective of tail latency for web service and adjusts the division of resources between the two partitions. Resource-usage monitoring is also used to tune resource allocation for particular resources.

1.6.3 Service-level provisioning in multi-tenant VM servers

When a server runs multiple VMs for different users, it is necessary to prevent one VM from using more resource than it has paid for and thereby prevent other tenants from being able to use the resource they have paid for. MPAM partitions provide a means to regulate the memory-system resources used by a VM.

While there need only be a few service levels provisioned onto a server, each VM needs a separate PARTID so that resource-usage controls can be separately responsive to the resource demands of that VM.

Chapter 2
MPAM and Arm Memory-System Architecture

This chapter contains the following sections:

2.1 MPAM and Arm memory-system architecture

This section is *normative*.

MPAM partitioning of memory-system performance resources must not affect the correctness of any memory behavior specified in the *Arm Architecture Reference Manual*. The Arm v8 memory model, as specified in that manual, must be followed in all of its particulars, including requirements for observation, coherence, caching, order, atomicity, endianess, alignment, memory types, and any other requirements defined in the Arm ARM v8 memory model. Furthermore, these requirements must also be met:

- For single-PE and multiple-PE environments.
- For MPAM information that is the same as, or different from, multiple requests by a single requestor or by multiple requestors.
- For all MPAM memory-system component resource controls and configurations.
- When MPAM information stored with data accessed from caches is the same as, or different from, MPAM information in requests that access that data.

A Speculative access (either an instruction prefetch or an early data read) may be generated at any time, based on MPAM System register configuration that might change before the access would be architecturally executed. MPAM does not impose any limit on such speculation – neither a data dependency on the MPAMn_ELx registers nor a control dependency on the System register synchronization, other than the limits on use of System register values in the *Arm Architecture Reference Manual*.

Chapter 3
ID Types, Properties, and Spaces

This chapter contains the following sections:

3.1 Introduction

This chapter is *normative*.

MPAM operation is based on the MPAM information that masters include with requests made to the memory system. This chapter defines the components of that MPAM information bundle, which consists of:

- PARTID
- PMG
- MPAM_NS

3.2 ID types and properties

MPAM has a single ID type, the Partition ID (PARTID). The architectural maximum width of a PARTID field is 16 bits.

PARTIDs have a single property, the Performance Monitoring Group (PMG). The architectural maximum width of a PMG field is 8 bits.

3.3 PARTID spaces and properties

MPAM PARTIDs have up to four PARTID spaces:

- Secure physical PARTID space. This space is accessed when a PE is executing in the Secure state.
- Non-secure physical PARTID space. This space is accessed when a PE is executing in the Non-secure state.
- Non-secure virtual PARTID space. This space exists only when the virtualization option is implemented and enabled for the current EL, and only within a PE that is executing in the Non-secure state.
- Secure virtual PARTID space. This space exists only when the virtualization option is implemented and enabled for the current EL, and only within a PE that is executing in the Secure state.

Each PARTID space has a maximum PARTID set by the implementation of the device. The range of valid PARTIDs is 0 to the maximum PARTID, inclusive. The maximum values of a PARTID implemented by a PE and by different MSCs need not be the same. Software should avoid using PARTIDs that exceed the smallest maximum of any MSCs accessed, because the behavior of an MSC accessed with an out-of-range PARTID is not well-defined.

Each MSC has an MPAM identification register with which to discover the width of PARTID implemented. The maximum Non-secure PARTID supported by an MSC is indicated in its MPAMF_IDR.PARTID_MAX. The maximum secure PARTID supported by an MSC is indicated in its MPAMF_SIDR.PARTID_MAX. The maximum PARTID supported by a PE is indicated in MPAMIDR_EL1.PARTID_MAX.

3.3.1 Default PARTID

Each MPAM PARTID space has a default value, which is PARTID 0 in that PARTID space.

The default physical PARTID must be generated when MPAM PARTID generation is disabled by MPAMEN == 0. This PARTID space is selected according to the current Security state; it is either the secure physical PARTID space or the Non-secure physical PARTID space.

MPAM PARTID generation is permitted to produce the default PARTID when the generation encounters an error.

3.3.2 Default PMG

The default PMG must be generated when MPAMEN == 0.

It is CONSTRAINED UNPREDICTABLE whether MPAM PMG generation produces the PMG value from the MPAMn_ELx register field or from the default PMG in either of two cases:

- When the PMG generation encounters an error, such as out-of-range PMG.
- When a default PARTID is generated due to a PARTID generation error.

In other cases, when MPAMEN == 1, the PMG must be the PMG value from the MPAMn_ELx register field.

The PARTID and PMG error conditions in a PE are described in *MPAM errors and default ID generation* on page 6-70.

System designers can also choose to output the default IDs on requests generated by masters that do not support MPAM.

3.4 Communication of MPAM information to MSCs

The representation of the MPAM information bundle on system interfaces is IMPLEMENTATION DEFINED.

If the system implements security, interfaces must be able to communicate a PARTID that is in either the Secure or the Non-secure PARTID space with each memory-system request.

3.5 MPAM_NS

MPAM_NS indicates the MPAM security space of the PARTID in the MPAM information bundle. In a PE, it is not stored in a register; it comes from the Security state. The pseudocode function, `IsSecure()`, computes the current Security state. See Table 6-5 on page 6-60 for combinations of MPAM_NS and NS bits.

Chapter 4
MSC Propagation of MPAM Information

This chapter contains the following sections:

4.1 Introduction

This section is *normative*.

MSC propagation of MPAM information includes the storage and transmission of:

- PARTID (*PARTID spaces and properties* on page 3-30).
- PMG (Chapter 10 *Resource Monitors*).
- MPAM_NS (*MPAM_NS* on page 3-32).

An MSC must implement at least one of the following categories of MPAM propagation behavior, and it might implement several of these behaviors.

4.2 Requesting master components

Requesting masters must label all requests to downstream MSCs with MPAM information.

A requesting master must have a device-appropriate means of setting the MPAM information in the request:

- The PE must use the scheme described in Chapter 6 *PE Generation of MPAM Information*.
- This architecture does not specify a mechanism for determining the MPAM information for requests from a non-PE master. Arm recommends that non-PE masters needing to use MPAM facilities specify a mechanism for determining the PARTID, PMG, and MPAM_NS of requests.
- SMMU architecture version 3.2 specifies MPAM information generation on memory system accesses translated by the SMMU and accesses originated by the SMMU to its tables in memory.
- GIC architecture version v3/v4 specifies MPAM information generation on memory system accesses originated by the GIC to its tables in memory.

If a requesting master does not support MPAM, the system must arrange to supply a value for MPAM information required for the interface. If no other mechanism is available, these values must be driven to a default value (in the Non-secure physical PARTID space or in the secure physical PARTID space).

See also *Requesters without MPAM support* on page 5-47.

4.3 Terminating slave components

A terminating slave receives requests from upstream masters but does not communicate the requests to a downstream slave. Instead, the terminating slave services the requests. A terminating slave does not forward MPAM information from a request. A terminating MSC is the edge of MPAM in a system.

A DRAM controller is a terminating slave, even though it communicates with DRAM devices to complete the request. The DRAM devices do not support MPAM communication, so MPAM information is not forwarded to them. This might also happen elsewhere in a system where there is no downstream slave that has MPAM support.

4.4 Intermediate slave-master components

Intermediate MSCs have both one or more slave interfaces and one or more master interfaces.

An intermediate component can route a request from an upstream master to one of its downstream master ports. When routing a request from upstream to downstream, the intermediate component passes the MPAM information unaltered to the downstream master port.

An intermediate component might terminate a request from upstream locally without propagating the request to a downstream master port if the request is serviced locally.

4.5 Request buffering

Requests can be buffered in any MSC. A request that is buffered must retain its MPAM information.

4.6 Cache memory

A cache line must store the MPAM information of the request that caused its allocation. See *Cache behavior* on page 5-50 for requirements on cache memory behavior.

Chapter 5
System Model

This chapter contains the following sections:

5.1 Introduction

This section describes a model of system behavior that can support the MPAM features. In particular, the behavior of masters, interconnects, caches, and memory controllers is described.

In this system model, a request:

- Begins at a requester (master), such as a PE, I/O master, DMA controller, or graphics processor:
  - MPAM information (PARTID, PMG, and MPAM_NS) is transported with every request.
- Traverses non-cache nodes that might be a transport component (such as an interconnect), a bus resizer, or an asynchronous bridge.
- Might reach an MSC that contains or is a cache:
  - Caches sometimes generate a response (cache hit) and sometimes pass the request on (cache miss).
  - Caches could also allocate entries based on the request.
  - Caches must store the MPAM PARTID, PMG, and MPAM_NS associated with an allocation:
    - Needed for cache-storage usage monitoring.
    - Used during eviction to another cache.
  - Cache eviction must attach MPAM fields to the eviction request. The source for MPAM information on an eviction may depend on whether the eviction is to memory or to another cache. See *Eviction* on page 5-50 and *Optional cache behaviors* on page 5-51.
- Might proceed from a cache to a transport component, and to other caches or a memory-channel controller.
- Might result in a memory controller or other terminating slave device responding to a request it receives.

Figure 5-1 on page 5-43 shows a simplified system model for the downstream flow, in the direction of requests from masters to slaves. In this figure, all objects implement an MSC except the PEs, I/O Masters, and I/O Slaves. PEs generate MPAM information from MPAM state in their System registers. I/O masters typically get their MPAM information when their requests pass through an SMMU.

The interconnects in Figure 5-1 on page 5-43 can represent bus, crossbar, packet, or other interconnect technologies.

An MSC responds to the MPAM information that arrives as part of a request. If the MSC implements partitioning controls, those controls find partitioning settings by the PARTID in the MPAM information of the request, and they use those settings to control the allocation of a controlled resource.

For caches, a cache line (which has an address) is always associated with the PARTID that allocated the line – or the PARTID that allocated the line into an inner cache that has now been evicted to the current cache. The inner cache PARTID must be preserved when the line is evicted to an outer cache.

An address may be accessed by multiple PARTIDs.

A cache must store the PARTIDs of the lines it contains, so that it can measure and control the cache lines used by a PARTID, and so that it can provide the PARTID to downstream MSCs when the line is evicted.

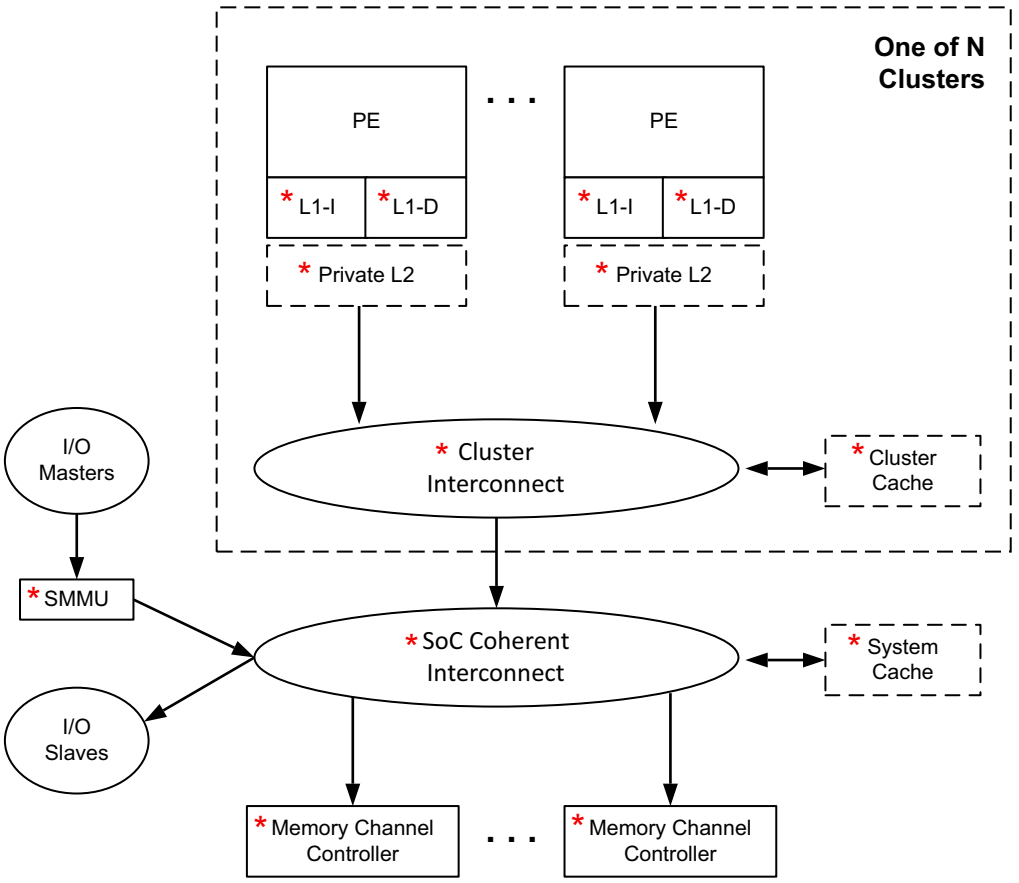


Figure 5-1 MPAM system model (downstream flow)

5.2 System-level field widths

Arm recommends that a system be configured to support a common size for the PARTID and PMG fields of MPAM. Mismatched sizes make it difficult for software to use anything but the smallest of implemented widths.

5.3 PE behavior

Processing elements (PEs) issue memory-system requests. PEs must implement the MPAMn_ELx registers (page 7-71) and their behaviors to generate the PARTID and PMG fields of memory-system requests.

See Chapter 6 *PE Generation of MPAM Information*.

5.3.1 PARTID generation

When a PE generates a memory-system request, it must label the request with the PARTID from the MPAMn_ELx register for the current Exception level. MPAM_NS must be set to the current execution Security state.

If the MPAM Virtualization Extension is implemented and enabled for the current Exception level, the PARTID from the MPAMn_ELx register must be mapped through the virtual partition mapping registers (*System register descriptions* on page 7-75) to produce a physical PARTID.

5.3.2 Information flow

When a PE with MPAM support issues a request to the rest of the system, it labels those commands with the PARTID and PMG supplied by software in the MPAMn_ELx register in effect (and if MPAM1_EL1 or MPAM0_EL1 with virtual PARTID mapping is enabled, with the virtual PARTID mapped to a physical PARTID).

In addition to the PARTID and PMG, the request must also have the MPAM_NS bit to indicate whether the PARTID is to be interpreted as in the Secure PARTID space or the Non-secure PARTID space.

5.3.3 Resource partitioning

If a PE contains internal memory-system partitioning controls, it must have memory-mapped registers (Chapter 9 *Resource Partitioning Controls*) to identify and configure those features.

The PE could include caches. The included caches could implement memory-system partitioning, such as cache-capacity partitioning controls. The cache behavior in *Cache behavior* on page 5-50 must apply to included cache functionality.

An MSC within a PE could have priority partitioning. This generates a priority or QoS value for the downstream traffic from that MSC, effectively giving priority or QoS values tied to the software environment that generated that traffic.

5.3.4 Resource-usage monitoring

A PE may have internal resource monitors that can measure the use by a PARTID and PMG of an MPAM resource (Chapter 10 *Resource Monitors*).

If a PE contains such features, they must have memory-mapped registers (Chapter 10 *Resource Monitors*) to identify and configure those features.

5.4 Other requesters with MPAM

Other requesters that support MPAM, such as a DMA controller, must issue requests to the system that have the MPAM fields. Non-PE masters can have schemes different from those implemented in PEs for associating MPAM information with requests. These other schemes are not documented herein.

5.5 Requesters without MPAM support

A requester that does not implement support for MPAM must use a system-specific means to provide MPAM information to MSCs that support MPAM.

Some examples of requester devices that might not implement support for MPAM include:

- Legacy DMA controller.
- Third-party peripheral IP.
- CoreSight DMA components, such as ETR.
- Older devices which cannot be economically upgraded to include MPAM support.

Some options for adding MPAM information to requests include:

- The MPAM information could be tied off to the default PARTID and PMG values (*Default PARTID* on page 3-30) and MPAM_NS set as appropriate for the device.
- The MPAM information could be provided by a System Memory Management Unit (SMMU) that supports adding MPAM information according to the stream and substream of the request.
- The MPAM information could be in added by a bus bridge or other system component that handles the requester's memory-system traffic.

Other implementations are permitted.

5.6 Model of a resource partitioning control

A general model of a resource partitioning controller within an MSC is shown in Figure 5-2. This model shows a resource partitioning model that measures resource usage by the partition and that controls resource usage by comparing the measured usage with the control settings for that partition.

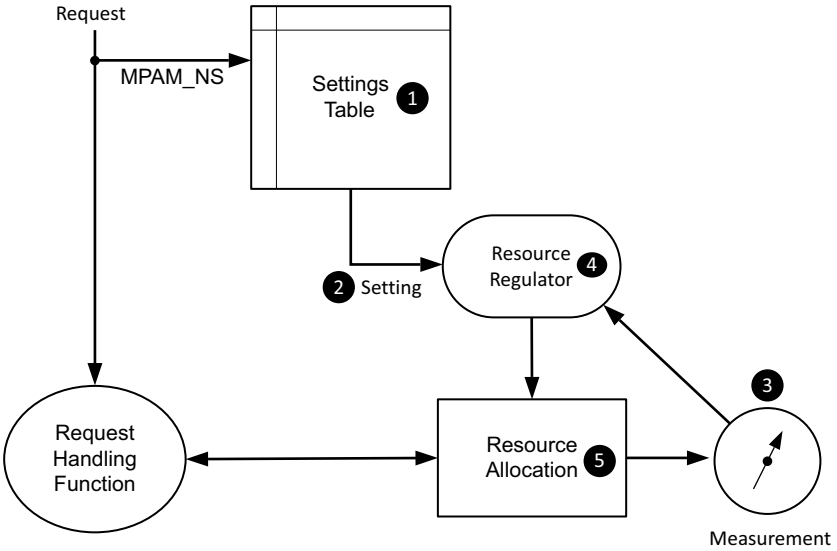


Figure 5-2 Model of MPAM resource partitioning controller

In Figure 5-2, a request arrives from an upstream master to an MSC that implements MPAM partitioning control. The request is handled as follows:

1. The PARTID and MPAM_NS values of the incoming request are used to index into a Settings Table of partition-control settings. (There is one settings table per implemented resource control.)
2. The table entry for that PARTID specifies its partition-control setting, which is passed to a Resource Regulator.
3. Conformance of the resource with the setting may require Measurement of how the resource is being used by the partition.
4. The Measurement feeds back to the Resource Regulator, where it is compared with the Setting and used to make a decision about Resource Allocation.

In Figure 5-2, items 1, 2, 3, and 4 are added to the original memory system when MPAM is implemented, although in some MSCs there may be sufficient measurement hardware already in place. Item 1, the Settings Table, is the heart of an MPAM MSC.

All of the above is separate from normal request-handling by the MSC.

When doing cache-way partitioning, a significant part of the above mechanism can be eliminated. It is not necessary to make measurements. The cache ways that can be allocated into are known.

The upside of cache-way partitioning is that it is simple and cheap. The downside is that caches do not have many ways, so fine-grained control is not possible. In addition, resources can be strained if one or more ways are allocated to only one partition, without sharing.

5.7 Interconnect behavior

Interconnects connect requesters to targets, and they must transport MPAM information fields from requester to target.

Interconnects may support the MPAM control features, such as priority partitioning. Support for MPAM is discoverable in ID registers and firmware data.

Some interconnect devices may include cache functionality, in which case the cache behavior in *Cache behavior* on page 5-50 applies.

5.8 Cache behavior

A cache must associate the MPAM information of the request that allocated a cache line with any data stored in the cache line. This stored MPAM information is a property of the data.

The term “data” in this section is intended to indicate the content stored in the cache. It is not intended to indicate any restriction on the applicability of this section based on the purpose of the cache or of its content.

The MPAM information on a request to the cache from an upstream master is used for the following purposes:

- Source for the MPAM information associated with data when the data is allocated into the cache and is stored in association with the data while the data resides in the cache.
- Optionally updating the stored MPAM information of the cached data on a store hit (*Write hits may update the MPAM information of a cache line* on page 5-51).
- Providing MPAM information for downstream requests to fulfill the incoming request such as a read from downstream on a cache miss that fetches data into the cache.
- Optionally (*Eviction*), providing MPAM information for downstream requests generated by evict or clean operations when this cache is the last level of cache upstream of main memory.
- Selecting settings of partitioning controls implemented in the cache.
- Tracking resource usage needed by partitions for a control implementation.
- Performing accounting, if necessary, to track resource usage for resource usage monitors, if implemented.
- Triggering and filtering resource monitors, if implemented, for events triggered by requests from upstream masters.

The stored MPAM information is used by MPAM for the following purposes:

- Providing the MPAM information for downstream requests generated by evict or clean operations, when this cache is not the last level of cache.
- Optionally (*Eviction*) providing MPAM information for downstream requests generated by evict or clean operations, when this cache is the last level of cache.
- Triggering and filtering resource monitors by MPAM PARTID and PMG, if implemented for events triggered by internal and downstream requests.
- Tracking resource usage by partitions, as needed by a partitioning control implementation.

5.8.1 Eviction

When a cache line is evicted to another cache, the evicting cache must produce the MPAM information that was used in the request that originally allocated the cache line.

A system cache (last-level cache) may produce the MPAM information of the request that caused the eviction in its request to a memory-channel controller, or the cache may produce the stored MPAM information associated with the evicted line.

5.8.2 Cache partitioning

A cache may optionally implement cache-partitioning resource controls, such as a cache-portion partitioning control.

The cache-portion partitioning control (*Cache-portion partitioning* on page 9-130) was conceived for use on large, multi-way associative caches, but cache-portion partitioning can be implemented on caches that are not set-associative. For example, a single entry or group of entries may be a cache portion in a fully-associative cache.

The cache maximum-capacity partitioning control (*Cache maximum-capacity partitioning* on page 9-131) was conceived for use on caches that do not support cache-portion partitioning or that have insufficient portions to meet the needs of the planned use.

Both types of cache partitioning may be used together in a cache memory component. This may be useful, for example, when the cache has insufficient portions to give adequate control for a planned use.

5.8.3 Resource monitoring

A cache may implement cache-storage usage monitoring (*Cache-storage usage monitor* on page 10-148). For a monitored PARTID, the monitor gives the total cache storage used by the PARTID.

5.8.4 Optional cache behaviors

The following cache behaviors are permitted but not required.

Write hits may update the MPAM information of a cache line

On a write hit to cached data that has different request MPAM information than the stored MPAM information associated with the data, the stored MPAM information is permitted to be updated to the request MPAM information.

It is possible that a change in the PART_ID of the data (without moving the data) leaves the data in a portion of the cache that the new PARTID does not have permission to allocate. This can occur if the Cache Portion Bit Map (CPBM) bit for that portion is not set in the CPBM for the new PARTID. The optional behavior in this subsection does not change the location within the cache, even if the new partition for the data does not have a CPBM bit that allows allocation in this portion of the cache. Updating the location within the cache is a second optional behavior that is covered in the next subsection.

Write hits that update the PARTID of a cache line may move that line to a different portion

A write hit to cached data is permitted to change the portion of the cache capacity allocated to the data, if (i) the PARTID of the cache data is updated due to the write hit, and (ii) the portion of capacity where the data currently resides is not in the new PARTID’s cache portion bitmap.

5.9 Memory-channel controller behavior

This section is informative.

A memory-channel controller may implement MPAM features. Some of the features that may be helpful in a memory-channel controller are:

- Memory-bandwidth minimum and maximum partitioning (*Memory-bandwidth minimum and maximum partitioning* on page 9-133).
- Memory-bandwidth portion partitioning (*Memory-bandwidth portion partitioning* on page 9-133).
- Priority partitioning (internal) (*Priority partitioning* on page 9-136).
- Memory-bandwidth usage monitors (*Memory-bandwidth usage monitors* on page 10-147).

Chapter 6
PE Generation of MPAM Information

This chapter contains the following sections:

6.1 Introduction

This introduction is *informative*. Other sections and subsections are *normative* unless marked as *informative*.

In a PE, the generation of PARTID, PMG, and MPAM_NS labels for memory-system requests is controlled by software running at the current EL or higher. The set of MPAM information for:

- An application running at EL0 is controlled from EL1.
- An OS or guest OS running at EL1 is controlled from EL1 or EL2, according to settings controlled at EL2 and EL3.
- A hypervisor or host OS running at EL2 is controlled from EL2 or EL3, according to settings controlled at EL3.
- A guest hypervisor running at EL1 is controlled from EL1 or EL2, according to settings controlled at EL2 and EL3.
- Secure instances of all of the above.
- Monitor software running at EL3 is controlled only from EL3.

6.2 Discovering MPAM

This section is *normative*.

The presence of MPAM functionality in a PE is indicated by the ID_AA64PFR0_EL1[43:40] field (Table 6-1) being non-zero. The version of the MPAM Extension architecture is indicated in this field.

Table 6-1 MPAM ID_AA64PFR0_EL1[43:40] values

ID_AA64PFR0_EL1[43:40]	MPAM Extension architecture version
`0x0`	MPAM Extension not present.
`0x1`	MPAM Extension is present with MPAM extension architecture 1.

If the MPAM Extension is present, MPAMIDR_EL1 specifies implementation properties, such as the largest PARTID usable and whether the virtualization features are supported.

6.3 MPAM System registers

This section is *normative*.

The MPAM PARTIDs are assigned to software by hypervisor and/or kernel software, and a PARTID, PMG, and MPAM_NS are associated with all memory-system requests originated by the PE.

The MPAMn_ELx System registers contain fields for two PARTIDs and the PMG property for each as shown in Table 6-2.

Table 6-2 MPAM System register PARTID and PMG fields

Field Name	Description
PARTID_D	PARTID used for data requests.
PARTID_I	PARTID used for instruction requests.
PMG_D	PMG property for PARTID_D.
PMG_I	PMG property for PARTID_I.

The MPAMn_ELx System registers use the register-name syntax shown in Figure 6-1. These registers control MPAM PARTID and PMG, as shown in Table 6-3 on page 6-57 and *Summary of system registers* on page 7-74 and *System register descriptions* on page 7-75.

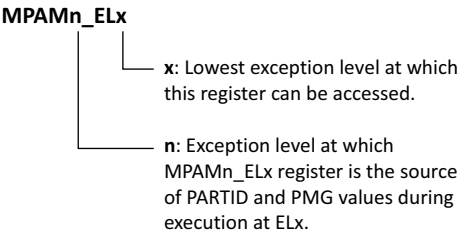


Figure 6-1 MPAM System register name syntax

Table 6-3 shows the PE MPAM System registers. The table does not include the following System registers: MPAMIDR_EL1, MPAMVPMn_EL2, MPAMVPMV_EL2, MPAMHCR_EL2.

Table 6-3 PE MPAM System registers

System register	Controlled From	Supplies PARTID and PMG when Executing In	Notes
MPAM0_EL1	EL3 EL2 EL1	EL0 (Applications)	With the virtualization option and MPAMHCR_EL2.EL0_VPMEN == 1, MPAM0_EL1 PARTIDs can be treated as virtual and mapped to a physical PARTID with virtualization option. Overridden by MPAM1_EL1 when MPAMHCR_EL2.GSTAPP_PLK is set. MPAM0_EL1 may be controlled from only EL3 if MPAM3_EL3.TRAPLOWER == 1, from only EL2 or EL3 if MPAM3_EL3.TRAPLOWER == 0 and MPAMHCR_EL2.TRAPMPAM0EL1 == 1 or from EL1, EL2 or EL3 if MPAM3_EL3.TRAPLOWER == 0 and MPAMHCR_EL2.TRAPMPAM0EL1 == 0.
MPAM1_EL1	EL3 EL2 EL1	EL1 (Guest OS)	Overrides MPAM0_EL1 when MPAMHCR_EL2.GSTAPP_PLK is set. With the virtualization option and MPAMHCR_EL2.EL1_VPMEN == 1, MPAM1_EL1 PARTIDs are treated as virtual and mapped to a physical PARTID. MPAM1_EL1 may be controlled only from EL3 if MPAM3_EL3.TRAPLOWER == 1, only from EL2 or EL3 if MPAM3_EL3.TRAPLOWER == 0 and MPAMHCR_EL2.TRAPMPAM1EL1 == 1, or from EL1, EL2 or EL3 if MPAM3_EL3.TRAPLOWER == 0 and MPAMHCR_EL2.TRAPMPAM1_EL1 == 0. When HCR_EL2.E2H == 1, accesses to MPAM1_EL1 through the MSR and MRS instructions are aliased to access MPAM2_EL2 instead.
MPAM2_EL2	EL3 EL2	EL2 (Hypervisor or host OS)	MPAM2_EL2 is controlled only from EL3 if MPAM3_EL3.TRAPLOWER == 1, or from EL2 or EL3 if MPAM3_EL3.TRAPLOWER == 0.
MPAM3_EL3	EL3	EL3 (Monitor)	MPAM3_EL3 is controlled only from EL3.
MPAM1_EL12	EL2	EL1	Alias to MPAM1_EL1 for type 2 hypervisor host executing with HCR_EL2.E2H == 1.

Table 6-4 on page 6-58 shows the selection of MPAMn_ELx System register for MPAM generation. All of the fields named are in MPAMHCR_EL2:

- GSTAPP_PLK is MPAMHCR_EL2.GSTAPP_PLK.
- EL0_VPMEN is MPAMHCR_EL2.EL0_VPMEN.
- EL1_VPMEN is MPAMHCR_EL2.EL1_VPMEN.

Table 6-4 Selection of MPAMn_ELx System register for MPAM generation

When current EL is:	Use PARTID and PMG fields from	Perform MPAM virtual PARTID mapping
EL0 with GSTAPP_PLK == 0	MPAM0_EL1	If EL0_VPMEN == 1
EL0 with GSTAPP_PLK == 1	MPAM1_EL1	If EL1_VPMEN == 1
EL1	MPAM1_EL1	If EL1_VPMEN == 1
EL2	MPAM2_EL2	Never
EL3	MPAM3_EL3	Never

6.4 Instruction, data, translation table walk, and other accesses

When a PE generates a memory-system request for an instruction access, the PARTID_I field of an MPAMn_ELx register is used, as shown in Table 6-4 on page 6-58. All translation table walk accesses for instructions use the same PARTID_I field that their instruction accesses use.

When a PE generates a memory-system request for a data access, the PARTID_D field of an MPAMn_ELx register is used, as shown in Table 6-4 on page 6-58. All translation table walk accesses for data access use the same PARTID_D field that their data accesses use.

PARTID_D and PARTID_I fields of an MPAMn_ELx register may be set by software to the same or different PARTIDs. If PARTID_D is used for an access, PMG_D from the same register must also be used. If PARTID_I is used for an access, PMG_I from the same register must also be used.

6.4.1 Load unprivileged and store unprivileged instructions

When executed at EL1 or at EL2 with EL2 Host (E2H), load unprivileged and store unprivileged instructions perform an access using permission-checking for an unprivileged access. These instructions do not change the MPAM labeling of the resulting memory-system requests from the labels that would be generated by other load or store instructions.

6.4.2 Accesses by enhanced support for nested virtualization

The Armv8.4 [4] enhanced support for nested virtualization turns MRS and MSR instructions to certain EL2 registers from EL1 into accesses to a data structure in EL2 memory. As such, these accesses generate PARTID and PMG using MPAM2_EL2.PARTID_D and MPAM2_EL2.PMG_D, respectively.

See *Support for nested virtualization* on page 6-67.

6.4.3 Accesses by statistical profiling extension

The Armv8.2 introduced the Statistical Profiling Extension (SPE). A PE with SPE can be configured to record statistically sampled events into a profiling buffer in memory. The buffer is accessed through the owning Exception level's translation regime.

MPAM PARTID, PMG, and MPAM_NS for SPE writes to the profiling buffer must use the SPE's owning Exception level MPAM data access values.

For example, if the owning Exception level is EL2, the profiling buffer writes must be performed with MPAM2_EL2.PARTID_D, MPAM2_EL2.PMG_D, and MPAM_NS reflecting the Security state of the owning Exception level.

6.5 Security

MPAM behavior in the PE and in MSCs is affected by the Security state. While the address spaces for Secure and Non-secure memory-system accesses are distinct, the memory-system resources are potentially shared in an implementation. For higher security, running Secure software with segregated resources can reduce the chances for side-channel attacks.

The generation of PARTID and PMG for a memory-system request is exactly the same at a Secure ELx as at a Non-secure ELx, for the same x. The difference is that requests at a Secure ELx are qualified with MPAM_NS to indicate Secure or Non-secure PARTID space.

MPAM security behavior in MSCs is covered in *Security in MSCs* on page 8-120.

6.5.1 Secure PARTID Space

In a PE, generation of Secure and Non-secure PARTIDs are governed by the following Secure MPAM PARTID space rules, described in *PARTID spaces and properties* on page 3-30:

- Secure PARTIDs are communicated with MPAM_NS == 0.
- Non-secure PARTIDs are communicated with MPAM_NS == 1.
- MPAM_NS == 0 if the current Security state is Secure.

In Secure execution, the sourcing of PARTID and PMG in a PE are as described in this specification for Non-secure execution. The PARTID and PMG generation uses MPAMn_ELx to source the labels for the request when executing at Exception level ELn. Non-secure and Secure PARTID generation is the same, including virtual-to-physical PARTID translation, if secure EL2 is present and enabled, and the MPAM virtualization feature is present and enabled for the MPAM0_EL1 or MPAM1_EL1 register used.

See also *PARTID virtualization* on page 6-61.

6.5.2 Non-secure MPAM PARTID space

Non-secure MPAM PARTIDs are communicated with MPAM_NS == 1.

Non-secure and Secure PARTID generation is the same, including virtual PARTID translation to physical PARTID if EL2 is present, the MPAM virtualization feature (MPAMIDR_EL1.HAS_HCR) is present, and virtual partition mapping is enabled for the MPAM0_EL1 or MPAM1_EL1 register used.

6.5.3 Indication of Secure MPAM PARTID spaces

Secure MPAM PARTIDs are communicated with MPAM_NS == 0.

MPAM_NS is distinct from the NS signal, which indicates the accessed address space is Non-secure or Secure. Table 6-5 shows the meaning of combinations of MPAM_NS and NS.

Table 6-5 Combinations of MPAM_NS and NS

MPAM_NS	NS	Meaning
0	0	Secure. A Secure PARTID (from Secure Execution state) used with access to a secure location.
0	1	Cross-state. A Secure PARTID (from Secure Execution state) used with access to a Non-secure location.
1	0	Non-secure PARTID (from Non-secure Execution state) used with access to Secure location. This is illegal because the MPAM_NS == 1 implies a Non-secure Execution state, which must never access a location in the Secure address space.
1	1	Non-secure PARTID (from Non-secure Execution state) used with access to a Non-secure location.

6.6 PARTID virtualization

This introduction to MPAM virtualization support is informative, but subsections are individually marked as *normative* or *informative*.

The PARTID virtualization features described in this section are only available in a Security state in which all of the following conditions are met:

- EL2 is implemented and enabled in the Security state. See also *Unimplemented exception levels* on page 7-114.
- MPAM virtualization is supported, as indicated by MPAMIDR_EL1.HAS_HCR == 1.

The hardware and software involved in supporting MPAM virtualization includes:

- Accesses made from EL1 to the MPAMIDR_EL1 register are trapped to EL2 under control of the MPAMHCR_EL2.TRAP_MPAMIDR_EL1 bit. This is done so that the hypervisor can emulate an MPAMIDR_EL1 access and present an altered view of the register to the guest OS running at EL1. This altered view shows that the PARTID_MAX field is a maximum that is equal to the largest virtual PARTID that the hypervisor has set up for the guest OS to use.
- Guest accesses to MPAM MSC control interfaces page-fault in the stage-2 page tables, thereby trapping to EL2 so that the virtual PARTID used can be access-controlled and mapped to the correct physical PARTID by the hypervisor. The hypervisor can give IPA mappings to an MSC's MPAM feature page that fault at stage 2 to produce this behavior.
- Mapping of guest OS-assigned virtual PARTID values into the physical PARTID space when running guest applications at EL0 and the guest OS at EL1.
- Optionally, an invalid virtual PARTID (that is, one in which the valid bit, MPAMVPMV_EL2, is 0) can cause a default virtual PARTID to be used. See *PARTID space on error* on page 6-70.
- Support for type-2 hypervisors (for example, kvm) with the HCR_EL2.E2H bit set when running the host OS in EL2 with hypervisor functionality. See *Support for type-2 hypervisors* on page 6-62.

These functions work together to give a guest OS the ability to control its virtual partitions and not trap to the hypervisor when context-switching between applications.

6.6.1 MPAM virtual ID spaces

This section is normative.

MPAM virtual ID spaces only exist if the MPAM virtualization option is implemented, as indicated in MPAMIDR_EL1.HAS_HCR.

Virtual PARTID spaces can be independently enabled for MPAM0_EL1 and MPAM1_EL1 in MPAMHCR_EL2. See Table 6-4 on page 6-58. These virtual spaces are mapped into physical PARTID spaces by MPAM virtual PARTID mapping System registers (MPAMVPM0_EL2 through MPAMVPM7_EL2) in PEs. The virtual PARTID mapping registers are set up from EL2 by the hypervisor.

When PARTID is being virtualized, the virtual PARTID is used to index an array of physical IDs contained in the virtual PARTID mapping registers. The index is also used to check the valid flag for that virtual PARTID mapping entry. If the virtual PARTID has a valid mapping, the physical PARTID from the selected virtual PARTID mapping register is used for the memory-system request.

If the virtual PARTID is greater than (4 * VPMR_MAX) + 3, it is outside of the range of virtual PARTID mapping register indices. An out-of-range virtual PARTID is permitted to be replaced by any other in-range virtual PARTID, and this replacement virtual PARTID is used to access the virtual PARTID mapping registers and valid bits. See *Example of virtual-to-physical PARTID mapping* on page 6-64.

If the virtual PARTID mapping entry accessed is invalid, the default virtual PARTID is used, if it is valid. If neither the accessed virtual PARTID mapping entry nor the default virtual PARTID mapping entry is valid, the default physical PARTID is used for the memory-system request. See *Default PARTID* on page 3-30.

6.6.2 Support for type-2 hypervisors

The beginning of this section is *normative*.

Arm introduced virtual host extensions in *Armv8.1 Extensions* to better support type-2 hypervisors, such as kvm. These extensions included the EL2 Host (E2H) bit in the hypervisor control register.

With type-2 hypervisors, the host runs at EL2 and runs host applications at EL0. The host runs guest OSs at EL1 with their applications at EL0. Type-2 hypervisors run with HCR_EL2.E2H == 1. In this case, some MPAM System register addresses access different MPAM System registers. This allows the host OS to run at EL2 while using the same System register addresses it would use when running at EL1.

Table 6-6 MPAM1_EL1 register accessed at EL2

System register accessing instruction	Named register	Associated register accessed at EL2
op1=0, CRn=10, CRm=5, op2=0	MPAM1_EL1	MPAM2_EL2

At EL2, accesses to an associated EL2 register using the normal (op1=4) encoding need explicit synchronization to be ordered with respect to accesses to the same register using this new mechanism.

In this configuration, the following aliases for the same set of EL1 registers are introduced for access at EL2 or EL3 (these registers are UNDEFINED at EL1 and EL0). A different register name is used to access the registers. When at EL3, accesses to the EL1 register using the normal (op1=0) value need explicit synchronization to be ordered with respect to accesses to the same register using this new mechanism.

Table 6-7 MPAM1_EL12 register accessed at EL2

System register accessing instruction	Named register	Associated register accessed at EL2
op1=5, CRn=10, CRm=5, op2=0	MPAM1_EL12	MPAM1_EL1

The remainder of this section is *informative*. It describes how a type-2 hypervisor (host OS) might use the MPAM hardware:

- MPAM1_EL12 is accessed by the host OS running at EL2 and is an alias for MPAM1_EL1. This register controls the MPAM PARTIDs and PMGs used when running a guest at EL1.
- MPAM1_EL1 is accessed by the host OS running at EL2 and is an alias for MPAM2_EL2. This register controls the host's access to its own MPAM controls.
- MPAM0_EL1 is accessed by the host OS running at EL2. This permits the host OS to control the MPAM PARTIDs and PMGs used by its applications. E2H does not alter this access. When running host applications at EL0, the host also sets HCR_EL2_TGE == 1 to route exceptions in the EL0 application to the host in EL2 rather than EL1.
- MPAMHCR_EL2 access is used by the host at EL2 to control the enables for virtual PARTID mapping and the trapping of MPAMIDR_EL1. E2H does not alter this access.
- MPAMVPMV_EL2 is used by the host at EL2 to control the validity of virtual PARTID mapping entries used to virtualize the guest's PARTIDs. E2H does not alter this access.
- MPAMVPMn_EL2 registers are used by the host at EL2 to contain the virtual PARTID mapping entries. These are set by the hypervisor at EL2 and used when running the guest OS and its applications. E2H does not alter this access.

The use of MPAM System registers by a guest OS is not altered by E2H:

- MPAM0_EL1 is accessed from EL1. This permits a guest OS to control the MPAM PARTIDs and PMGs used by its applications. E2H does not alter this access.

- MPAM1_EL1 is accessed by the guest OS running at EL1 to change MPAM context for the guest OS running at EL1, unless trapped to EL2 by MPAM2_EL2.TRAPMPAM1EL1 == 1, or trapped to EL3 by MPAM3_EL3.TRAPLOWER == 1. E2H does not alter this access.

6.6.3 Mapping of guest OS virtual PARTIDs

This section is *informative*. It describes how software might use MPAM hardware.

When virtualizing MPAM, the hypervisor controls the use of PARTIDs by guest OSs. The hypervisor can:

- Set the number of virtual PARTIDs that a guest OS is permitted to assign and use. This number is communicated by trapping access by the guest to MPAMIDR_EL1.
- Permit the guest OS to use virtual PARTIDs for applications running at EL0 and to change them by writing to MPAM0_EL1.
- Permit the guest OS to also use virtual PARTIDs when running at EL1 and to change them by writing to MPAM1_EL1.
- Map each of the guest's virtual PARTIDs from the range of 0 to the maximum guest PARTID into a physical PARTID for the current Security state. It does this by means of the MPAMVPMn virtual PARTID mapping registers that are managed by the hypervisor.

PMGs modify PARTID and do not need any further virtualization support.

Virtualized guests are limited to using PARTIDs in the range of 0 to n, where n is the implemented virtual PARTID mapping entries. The parameters are:

- MPAMIDR_EL1.VPMR_MAX has the number of virtual PARTID mapping registers implemented. Each virtual PARTID mapping register contains four mapping entries.
- The largest virtual PARTID is n = (4 * VPMR_MAX) + 3.

If VPMR_MAX == 0, there is only one virtual PARTID mapping register, 4 virtual PARTID mapping entries, and the maximum corresponding virtual PARTID is 3.

The following registers and fields are used to control virtualization:

MPAMHCR_EL2 control fields:

- EL0_VPMEN: Enable virtual PARTID mapping from MPAM0_EL1 when executing an application at EL0. If HCR_EL2.E2H == 1 and HCR_EL2.TGE == 1, MPAM is not virtualized EL0. If GSTAPP_PLK == 1, MPAM1_EL1 is used instead of MPAM0_EL1 when executing at EL0 and virtualization of PARTIDs is controlled by EL1_VPMEN.
- EL1_VPMEN: Enable virtual PARTID mapping from MPAM1_EL1 when executing a guest OS at EL1. If GSTAPP_PLK == 1 when executing at EL0, MPAM1_EL1 is used instead of MPAM0_EL1 and MPAM virtualization is controlled by EL1_VPMEN instead of EL0_VPMEN.

MPAMVPM0_EL2 to MPAMVPM7_EL2 registers:

- Each register has four 16-bit fields. Each field contains a physical PARTID.
- Together they form a virtual PARTID mapping vector that maps the virtual PARTIDs into the physical PARTID space.
- Within each physical PARTID field, only sufficient low-order bits are required to represent the MPAMIDR_EL1.PARTID_MAX. Higher-order bits may be implemented as RAZ/WI.

MPAMVPMV_EL2 register:

- MPAMVPMV_EL2 contains 4*(m+1) valid bits, indexed from 0 to (4*m + 3), one bit for each of the implemented virtual PARTIDs supported in the MPAMVPMn_EL2 registers, where m = MPAMIDR_EL1.VPMR_MAX and n ranges from 0 to n.

- There can be up to 32 virtual-to-physical PARTID mappings. If a virtual PARTID is greater than the maximum index supported, an in-range virtual PARTID is permitted to be accessed instead (*MPAM AArch32 interoperability* on page 6-66).

Example of virtual-to-physical PARTID mapping

This section is *informative*.

- If the current execution level is EL1:
  - If EL1_VPMEN == 0, then virtualization is disabled at EL1, and MPAM1_EL1.PARTID_D and MPAM1_EL1.PARTID_I are physical PARTIDs.
  - If EL1_VPMEN == 1, then virtualization is enabled at EL1 and MPAM1_EL1.PARTID_D and MPAM1_EL1.PARTID_I are virtual PARTIDs that are to be mapped to physical PARTIDs.
- Assume MPAMIDR_EL1.VPMR_MAX == 0b010. That means the largest virtual PARTID is 4*2+3 = 11. Therefore, 12 virtual PARTIDs, from 0 to 11, can be mapped to physical PARTIDs.
- Assume MPAM1_EL1.PARTID_D contains 6:
  - MPAMVPMV_EL2.VPM_V<6> is checked to determine if the mapping for virtual PARTID 6 is valid. MPAMVPMV_EL2.VPM_V<6> == 1 means virtual PARTID 6 is valid. MPAMVPMV_EL2.VPM_V<6> == 0 means virtual PARTID 6 is invalid.
  - If a valid mapping exists (VPM_V<6> == 1), the physical PARTID is in MPAMVPM1_EL2.Phys_PARTID6.
  - If a valid mapping does not exist (VPM_V<6> == 0), the mapping for the default virtual PARTID is used.

    If a valid mapping does not exist for the default virtual PARTID, the default physical PARTID is used.
- For out-of-range virtual PARTIDs, an implementation can choose any other virtual PARTID to use instead. This permits truncation of inputs that have too many bits. It also permits other reductions to in-range PARTIDs. For example, if VPMR_MAX is 2, the virtual PARTID 13 is out of range. In this example, an implementation might save time by forcing the 8s bit (bit number 4) to 0 when both the 8s bit and 4s bit (bit number 3) are 1 in the virtual PARTID. This technique selects virtual PARTID mapping entry 5 instead of out-of-range 13. The technique is sometimes called “replacement virtual PARTID”. One must still do the steps of bullet 3, above, on the replacement virtual PARTID.

6.6.4 Guest OS and all its applications under single PARTID

This section is *normative*.

GSTAPP_PLK is a control bit in MPAMHCR_EL2. The bit causes MPAM1_EL1 to be used instead of MPAM0_EL1 when executing at EL0. This GSTAPP_PLK function runs all EL0 applications of a VM in the same partition as the EL1 guest OS.

When GSTAPP_PLK is active, MPAM0_EL1 is not used for PARTID or PMG generation. If virtual PARTID mapping is enabled for EL1, the EL1 PARTID_I or PARTID_D is mapped to a physical PARTID before being used for requests originating from applications at EL0, as well as for the guest OS at EL1.

Note

The guest OS at EL1 cannot determine whether GSTAPP_PLK is active or not. EL1 access to read and write MPAM0_EL1 is not affected by GSTAPP_PLK == 1.

6.6.5 Trap accesses to EL2 and EL1 System registers

The available traps include those that:

- Virtualize MPAMIDR_EL1.
- Control access by EL1 to MPAM1_EL1 and MPAM0_EL1.

- Control access by EL3 to MPAM registers from lower ELs.

Virtualizing MPAMIDR_EL1

EL2 software can force accesses to MPAMIDR_EL1 to trap to EL2 by setting MPAMHCR_EL2.TRAP_MPAMIDR_EL1 == 1. By trapping MPAMIDR_EL1, an EL2 hypervisor can provide an emulated value for MPAMIDR_EL1 to the EL1 software.

Trapping accesses to MPAM2_EL2

Accesses to MPAM2_EL2 from EL2 are trapped to EL3 when MPAM3_EL3.TRAPLOWER == 1.

Controlling accesses to MPAM1_EL1

EL2 software can control whether EL1 software can access MPAM1_EL1. Accesses to MPAM1_EL1 from EL1 are trapped to EL2 when MPAM2_EL2.TRAPMPAM1EL1 == 1.

MPAM1_EL12 is an alias for MPAM1_EL1 accessed from EL2. It is therefore not subject to traps from MPAM2_EL2.TRAPMPAM1EL1.

When HCR_EL2.E2H == 1, MPAM1_EL1 is an alias for MPAM2_EL2 accessed from EL2. It is therefore not subject to traps from MPAM2_EL2.TRAPMPAM1EL1.

Controlling accesses to MPAM0_EL1

EL2 software can control whether EL1 software can access MPAM0_EL1. Accesses to MPAM0_EL1 from EL1 are trapped to EL2 when MPAM2_EL2.TRAPMPAM0EL1 == 1.

Trapping all MPAM registers

When EL2 or EL1 software does not context switch MPAM state, such as when the software does not support MPAM at all, the MPAM System registers might be used to pass information between virtual machines or applications.

EL3 software can trap accesses to MPAM registers from lower Exception levels to EL3 by setting MPAM3_EL3.TRAPLOWER == 1.

TRAPLOWER protects against misuse of the MPAM state registers when EL2 software is does not support MPAM context switching.

If EL2 software is present and supports MPAM but EL1 software does not, MPAM2_EL2.TRAPMPAM1EL1 and TRAPMPAM0EL1 protect against misuse by an unaware guest while permitting EL2 to set up an MPAM environment for that guest.

If there is no EL2 or no EL2 software, TRAPLOWER can prevent misuse of MPAM registers by EL1 software.

MPAM3_EL3.TRAPLOWER traps have priority over all traps controlled by MPAM2_EL2 and MPAMHCR_EL2.

6.7 MPAM AArch32 interoperability

This section is *normative*.

MPAM System registers are not accessible from AArch32. If a lower EL uses AArch32, its MPAM PARTIDs and PMGs are controlled by a higher level that uses AArch64.

6.8 Support for nested virtualization

This section is *normative*.

Armv8.3 Extensions added architectural support for nested virtualization, and *Armv8.4 Extensions* added enhancements to the nested virtualization support. This section describes the support of MPAM with these extensions.

6.8.1 Nested virtualization extension

Armv8.3 Extensions added support for nested virtualization. This subsection only applies if Armv8.3 nested virtualization extension is implemented.

Table 6-8 lists the System registers that are trapped from EL1 to EL2 rather than being UNDEFINED when HCR_EL2.NV == 1, and HCR_EL2.NV2 == 0, and MPAM3_EL3.TRAPLOWER == 0.

Table 6-8 Registers trapped from EL1 to EL2 when HCR_EL2.NV == 1

MPAM1_EL12	MPAMVPMV_EL2	MPAMVPM2_EL2	MPAMVPM5_EL2
MPAM2_EL2	MPAMVPM0_EL2	MPAMVPM3_EL2	MPAMVPM6_EL2
MPAMHCR_EL2	MPAMVPM1_EL2	MPAMVPM4_EL2	MPAMVPM7_EL2

When HCR_EL2.NV == 1, and HCR_EL2.NV2 == 0, and MPAM3_EL3.TRAPLOWER == 1, access to any of the listed MPAM System registers from EL1 traps to EL3.

There are no other changes to the v8.3 nested virtualization extension to support the MPAM Extension.

6.8.2 Enhanced nested virtualization extension

Armv8.4 Extensions introduced an enhancement for nested virtualization. This enhancement transforms direct reads or writes (the terms “direct reads” and “direct writes” are defined in the Arm ARM) of several registers (that is, the target System register names in an MRS or MSR instruction) from EL1 to loads or stores, respectively, in the same Security state.

The remainder of this section applies only if both Armv8.3 nested virtualization extension and Armv8.4 enhanced nested virtualization extension are implemented.

If HCR_EL2.NV2 == 0, MSR or MRS instructions do not cause reads or writes to occur to the memory, and the behavior of the HCR_EL2.NV and HCR_EL2.NV1 bits is as described in the Armv8.3 architecture.

If HCR_EL2.NV2 == 1:

- If HCR_EL2.NV == 1 and HCR_EL2.NV1 ==0 for a Security state, direct reads or writes of any of the following MPAM register names (that is, the target System register names in the MRS or MSR instruction) from EL1 in the same Security state to be treated as loads or stores respectively. The memory address access is VNCR_EL2.BADDR<<12 + Offset from Table 6-9 as described in *Armv8.4 Extensions*.

Table 6-9 Enhanced nested virtualization offsets of System registers (NV2 == 1, NV1 == 0, and NV ==1)

Register Name	Offset
MPAM1_EL12	0x900
MPAMHCR_EL2	0x930
MPAMVPMV_EL2	0x938
MPAMVPM0_EL2	0x940
MPAMVPM1_EL2	0x948
MPAMVPM2_EL2	0x950
MPAMVPM3_EL2	0x958
MPAMVPM4_EL2	0x960
MPAMVPM5_EL2	0x968
MPAMVPM6_EL2	0x970
MPAMVPM7_EL2	0x978

- If HCR_EL2.NV == 1 and HCR_EL2.NV1 == 1 for a Security state, direct reads or writes of any of the registers in Table 6-10 (that is, the target System register names in an MRS or MSR instruction) from EL1 in the same Security state are treated as loads or stores, respectively, in the same Security state. The memory address access is VNCR_EL2.BADDR<<12 + Offset from Table 6-8 on page 6-67 as described in *Armv8.4 Extensions*.

Table 6-10 Enhanced nested virtualization offsets of System registers (NV2 == 1, NV1 == 1 and NV == 1)

Register Name	Offset
MPAM1_EL1	0x900
MPAMHCR_EL2	0x930
MPAMVPMV_EL2	0x938
MPAMVPM0_EL2	0x940
MPAMVPM1_EL2	0x948
MPAMVPM2_EL2	0x950
MPAMVPM3_EL2	0x958
MPAMVPM4_EL2	0x960
MPAMVPM5_EL2	0x968
MPAMVPM6_EL2	0x970
MPAMVPM7_EL2	0x978

When HCR_EL2.NV == 1 and HCR_EL2.NV2 == 1, MPAM3_EL3.TRAPLOWER is overridden for those registers listed in Table 6-9 if HCR_EL2.NV1 == 0 or in Table 6-10 if HCR_EL2.NV1 == 1. When HCR_EL2.NV == 1 and HCR_EL2.NV2 == 1, MPAM3_EL3.TRAPLOWER == 1 does not cause an access from EL1 to an MPAM System register in the tables to be trapped to EL3, but that access is converted to a memory read or write as described in this subsection.

6.8.3 MPAM PARTID and PMG for enhanced nested virtualization loads and stores

For Armv8.4 enhanced nested virtualization support, when HCR_EL2.NV2 == 1 and HCR_EL2.NV == 1, MRS or MSR instructions to any System register that are converted to loads or stores must be performed with the MPAM PARTID_D and PMG_D from MPAM2_EL2.

6.9 MPAM errors and default ID generation

MPAM errors are detected when a memory request is generated by a load, store, fetch, or table-walk with the following conditions:

- Physical or virtual PARTID or PMG is out of range.
- Virtual PARTID n is invalid, as indicated by MPAMVPMV_EL2<n>.

In a given implementation, some errors may never occur. For example, an implementation with only w bits of PARTID and MPAMIDR.PARTID_MAX as (2w – 1), and that truncates PARTID values with non-zero bits higher than w – 1, can never have a physical PARTID out-of-range error. See *Default PARTID* on page 3-30.

6.9.1 Out-of-range PARTID behavior

The behavior of a PE when a physical or virtual PARTID from PARTID_I or PARTID_D of an MPAMn_ELx register is out of range is CONSTRAINED UNPREDICTABLE as one of:

- The out-of-range PARTID is replaced by the default PARTID in the same PARTID space.
- The out-of-range PARTID is replaced by any in-range PARTID in the same PARTID space.

6.9.2 Out-of-range PMG behavior

The behavior of a PE when an MPAMn_ELx register's PMG_I or PMG_D is out-of-range CONSTRAINED UNPREDICTABLE is one of:

- The out-of-range PMG is replaced by the default PMG.
- The out-of-range PMG is replaced by any in-range PMG.

6.9.3 Invalid virtual PARTID behavior

The behavior of a PE, when (i) a PARTID_I or PARTID_D from an MPAMn_ELx register (or a replacement PARTID as in *Out-of-range PARTID behavior*) is used as a virtual PARTID n, and (ii) the corresponding bit MPAM_VMPV_EL2<n> == 0, the default virtual PARTID must be used if it is valid (MPAM_VPMV_EL2<0> == 1). If neither the accessed virtual PARTID mapping entry nor the default virtual PARTID mapping entry is valid, the default physical PARTID must be used for the memory-system request. See *Default PARTID* on page 3-30.

6.9.4 PARTID space on error

When an error is encountered in the generation of PARTID, the replacement PARTID is generated in the PARTID space as shown in Table 6-11.

Table 6-11 PARTID space for PARTID generation errors

Error	Space of replacement PARTID
NS virtual PARTID out of range	NS virtual PARTID
NS virtual PARTID mapping entry invalid	NS virtual PARTID
NS default virtual PARTID is invalid	NS physical PARTID
S virtual PARTID out of range	S virtual PARTID
S virtual PARTID mapping entry invalid	S virtual PARTID
NS physical PARTID out of range	NS physical PARTID
S physical PARTID out of range	S virtual PARTID

Chapter 7
System Registers

This chapter contains the following sections:

7.1 Overview

System registers are implemented in PEs and accessed using the Move System Register to General-Purpose Register (MRS) and Move General-Purpose Register To System Register (MSR) instructions.

7.2 Synchronization of system register changes

Direct writes to system registers are only guaranteed to be visible to indirect reads (such as using the MPAM system registers to generate MPAM information for memory requests) after a Context synchronization event, as described in the Arm Architecture Reference Manual for Armv8-A [1].

MPAM system register writes must be visible for generation of MPAM information in new memory requests after a Context synchronization event.

When MPAM system registers are set at one EL and used for generation of MPAM information at another EL, the change of EL is a Context synchronization event that makes the previous direct writes to MPAM registers visible for generating MPAM information.

If an MPAM register is updated at the same EL at which it is used for generation of MPAM information on memory-system requests, software must ensure that a Context synchronization event, such as an Instruction Synchronization Barrier (ISB), is executed after the direct write to the MPAM system register and before the changed system register value is certain to be used for labeling memory-system requests.

The Arm Architecture Reference Manual for Armv8-A [1] requires that a direct write to a system register must not affect instructions before the direct system-register write in program order.

If system registers are used for configuration of memory-system controls that are implemented in the PE, a Data Synchronization Barrier (DSB) must ensure that the prior memory accesses are completed before the update. No such system registers are defined herein. Additional requirements will be described if and when such requirements are added.

When MPAM system registers are updated, TLB maintenance is not required. Only a Context synchronization event is required before the updated value is guaranteed to be used for memory requests. Thus, MPAM information is not permitted to be cached in a TLB and used in lieu of using system registers for the generation of MPAM information.

7.3 Summary of system registers

In a PE, the MPAM system registers shown in Table 7-1 control the generation of PARTID and PMG by the PE, according to the EL and configuration of MPAM. See *Discovering MPAM* on page 6-55.

Table 7-1 Summary of system registers

op1	CRn	CRm	op2	System register	Description
0	10	5	1	MPAM0_EL1	MPAM context for EL0 execution.
0	10	5	0	MPAM1_EL1	MPAM context for EL1 execution.
4	10	5	0	MPAM2_EL2	MPAM context for EL2 execution.
6	10	5	0	MPAM3_EL3	MPAM context for EL3 execution.
5	10	5	0	MPAM1_EL12	MPAM context for EL1 execution on type-2 hypervisor.
4	10	4	0	MPAMHCR_EL2	Hypervisor configuration register for virtualization of PARTID in EL0.
4	10	4	1	MPAMVPMV_EL2	Virtual PARTID map valid bits.
4	10	6	0-7	MPAMVPM0_EL2 through MPAMVPM7_EL2	Virtual PARTID mapping for virtualization.
0	10	4	4	MPAMIDR_EL1	MPAM identification register.

7.4 System register descriptions

This section lists the MPAM System registers in AArch64.

7.4.1 MPAM0_EL1, MPAM0 Register (EL1)

The MPAM0_EL1 characteristics are:

Purpose

Holds information to generate MPAM labels for memory requests when executing at EL0. When EL2 is present and enabled, the MPAM virtualization option is present, MPAMHCR_EL2.GSTAPP_PLK == 1 and HCR_EL2.TGE == 0, MPAM1_EL1 is used instead of MPAM0_EL1 to generate MPAM information to label memory requests.

If EL2 is present and enabled, and HCR_EL2.E2H == '0' or HCR_EL2.TGE == '0', the MPAM virtualization option is present and MPAMHCR_EL2.EL0_VPMEN == 1, MPAM PARTIDs in MPAM0_EL1 are virtual and mapped into physical PARTIDs for the current Security state.

Configurations

This register is present only when MPAM is implemented. Otherwise, direct accesses to MPAM0_EL1 are UNDEFINED.

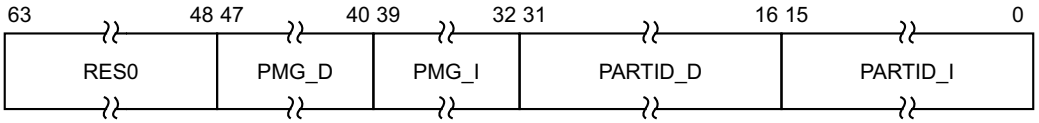
RW fields in this register reset to architecturally UNKNOWN values.

Attributes

MPAM0_EL1 is a 64-bit register.

Field descriptions

The MPAM0_EL1 bit assignments are:



Bits [63:48]

Reserved, RES0.

PMG_D, bits [47:40]

Performance monitoring group property for PARTID_D.

This field resets to an architecturally UNKNOWN value.

PMG_I, bits [39:32]

Performance monitoring group property for PARTID_I.

This field resets to an architecturally UNKNOWN value.

PARTID_D, bits [31:16]

Partition ID for data accesses, including load and store accesses, made from EL0.

This field resets to an architecturally UNKNOWN value.

PARTID_I, bits [15:0]

Partition ID for instruction accesses made from EL0.

This field resets to an architecturally UNKNOWN value.

Accessing the MPAM0_EL1

None of the fields in this register are permitted to be cached in a TLB.

Accesses to this register use the following encodings in the System instruction encoding space:

MRS <Xt>, MPAM0_EL1

op0	op1	CRn	CRm	op2
0b11	0b000	0b1010	0b0101	0b001

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MPAM2_EL2.TRAPMPAM0EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        return MPAM0_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return MPAM0_EL1;
elsif PSTATE.EL == EL3 then
    return MPAM0_EL1;
```

MSR MPAM0_EL1, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b000	0b1010	0b0101	0b001

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MPAM2_EL2.TRAPMPAM0EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        MPAM0_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        MPAM0_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    MPAM0_EL1 = X[t];
```

7.4.2 MPAM1_EL1, MPAM1 Register (EL1)

The MPAM1_EL1 characteristics are:

Purpose

MPAM1_EL1 holds information to generate MPAM labels for memory requests when executing at EL1.

When EL2 is present and enabled, the MPAM virtualization option is present, MPAMHCR_EL2.GSTAPP_PLK == 1 and HCR_EL2.TGE == 0, MPAM1_EL1 is used instead of MPAM0_EL1 to generate MPAM labels for memory requests when executing at EL0.

MPAM1_EL1 is an alias for MPAM2_EL2 when executing at EL2 with HCR_EL2.E2H == 1.

MPAM1_EL12 is an alias for MPAM1_EL1 when executing at EL2 or EL3 with HCR_EL2.E2H == 1.

If EL2 is is present and enabled, the MPAM virtualization option is present and MPAMHCR_EL2.EL1_VPMEN == 1, MPAM PARTIDs in MPAM1_EL1 are virtual and mapped into physical PARTIDs for the current Security state. This mapping of MPAM1_EL1 virtual PARTIDs to physical PARTIDs when EL1_VPMEN is 1 also applies when MPAM1_EL1 is used at EL0 due to MPAMHCR_EL2.GSTAPP_PLK.

Configurations

AArch64 System register MPAM1_EL1[63] is architecturally mapped to AArch64 System register MPAM3_EL3[63] when HaveEL(EL3).

AArch64 System register MPAM1_EL1[63] is architecturally mapped to AArch64 System register MPAM2_EL2[63] when !HaveEL(EL3) and HaveEL(EL2).

This register is present only when MPAM is implemented. Otherwise, direct accesses to MPAM1_EL1 are UNDEFINED.

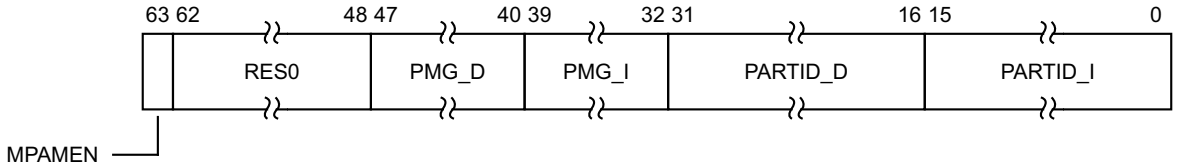
Some or all RW fields of this register have defined reset values. These apply only if the PE resets into an Exception level that is using AArch64. Otherwise, RW fields in this register reset to architecturally UNKNOWN values.

Attributes

MPAM1_EL1 is a 64-bit register.

Field descriptions

The MPAM1_EL1 bit assignments are:



MPAMEN, bit [63]

MPAM Enable: MPAM is enabled when MPAMEN == 1. When disabled, all PARTIDs and PMGs are output as their default value in the corresponding ID space.

Values of this field are:

`0b0` The default PARTID and default PMG are output in MPAM information.

`0b1` MPAM information is output based on the MPAMn_ELx register for ELn according the MPAM configuration.

If neither EL3 nor EL2 is implemented, this field is read/write.

If EL3 is implemented, this field is read-only and reads the current value of the read/write bit MPAM3_EL3.MPAMEN.

If EL3 is not implemented and EL2 is implemented, this field is read-only and reads the current value of the read/write bit MPAM2_EL2.MPAMEN.

This field resets to `0`.

Accessing this field has the following behavior:

- When !HaveEL(EL3) and !HaveEL(EL2), access to this field is RW.
- Otherwise, access to this field is RO.

Bits [62:48]

Reserved, RES0.

PMG_D, bits [47:40]

Performance monitoring group property for PARTID_D.

This field resets to an architecturally UNKNOWN value.

PMG_I, bits [39:32]

Performance monitoring group property for PARTID_I.

This field resets to an architecturally UNKNOWN value.

PARTID_D, bits [31:16]

Partition ID for data accesses, including load and store accesses, made from EL1.

This field resets to an architecturally UNKNOWN value.

PARTID_I, bits [15:0]

Partition ID for instruction accesses made from EL1.

This field resets to an architecturally UNKNOWN value.

Accessing the MPAM1_EL1

When HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL3 using the mnemonic MPAM1_EL1 or MPAM1_EL12 are not guaranteed to be ordered with respect to accesses using the other mnemonic.

None of the fields in this register are permitted to be cached in a TLB.

Accesses to this register use the following encodings in the System instruction encoding space:

MRS <Xt>, MPAM1_EL1

op0	op1	CRn	CRm	op2
0b11	0b000	0b1010	0b0101	0b000

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MPAM2_EL2.TRAPMPAM1EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '111' then
        return NVMem[0x900];
    else
        return MPAM1_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
```

```
    elsif HCR_EL2.E2H == '1' then
        return MPAM2_EL2;
    else
        return MPAM1_EL1;
elsif PSTATE.EL == EL3 then
    return MPAM1_EL1;
```

MSR MPAM1_EL1, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b000	0b1010	0b0101	0b000

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MPAM2_EL2.TRAPMPAM1EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '111' then
        NVMem[0x900] = X[t];
    else
        MPAM1_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HCR_EL2.E2H == '1' then
        MPAM2_EL2 = X[t];
    else
        MPAM1_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    MPAM1_EL1 = X[t];
```

MRS <Xt>, MPAM1_EL12

op0	op1	CRn	CRm	op2
0b11	0b101	0b1010	0b0101	0b000

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
        return NVMem[0x900];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        if HaveEL(EL3) && !ELUsingAArch32(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
        if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return MPAM1_EL1;
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
```

```
        return MPAM1_EL1;
    else
        UNDEFINED;
```

MSR MPAM1_EL12, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b101	0b1010	0b0101	0b000

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
        NVMem[0x900] = X[t];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        if HaveEL(EL3) && !ELUsingAArch32(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
        if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            MPAM1_EL1 = X[t];
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
        MPAM1_EL1 = X[t];
    else
        UNDEFINED;
```

7.4.3 MPAM2_EL2, MPAM2 Register (EL2)

The MPAM2_EL2 characteristics are:

Purpose

Holds information to generate MPAM labels for memory requests when executing at EL2.

Configurations

AArch64 System register MPAM2_EL2[63] is architecturally mapped to AArch64 System register MPAM3_EL3[63] when HaveEL(EL3).

AArch64 System register MPAM2_EL2[63] is architecturally mapped to AArch64 System register MPAM1_EL1[63].

This register is present only when MPAM is implemented. Otherwise, direct accesses to MPAM2_EL2 are UNDEFINED.

This register has no effect if EL2 is not enabled in the current Security state.

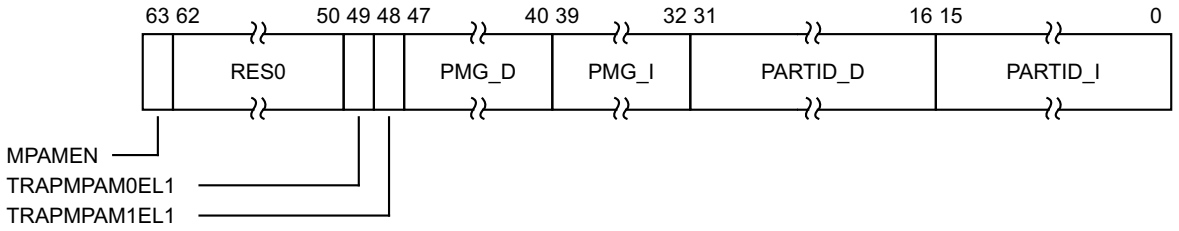
Some or all RW fields of this register have defined reset values. These apply only if the PE resets into an Exception level that is using AArch64. Otherwise, RW fields in this register reset to architecturally UNKNOWN values.

Attributes

MPAM2_EL2 is a 64-bit register.

Field descriptions

The MPAM2_EL2 bit assignments are:



MPAMEN, bit [63]

MPAM Enable: MPAM is enabled when MPAMEN == 1. When disabled, all PARTIDs and PMGs are output as their default value in the corresponding ID space.

Values of this field are:

`0b0` The default PARTID and default PMG are output in MPAM information from all ELx.

`0b1` MPAM information is output based on the MPAMn_ELx register for ELn according the MPAM configuration.

If EL3 is not implemented, this field is read/write.

If EL3 is implemented, this field is read-only and reads the current value of the read/write MPAM3_EL3.MPAMEN bit.

This field resets to `0`.

Accessing this field has the following behavior:

- When !HaveEL(EL3), access to this field is RW.
- Otherwise, access to this field is RO.

Bits [62:50]

Reserved, RES0.

TRAPMPAM0EL1, bit [49]

TRAPMPAM0EL1: Trap accesses from EL1 to the MPAM0_EL1 register trap to EL2.

0b0 Accesses to MPAM0_EL1 from EL1 are not trapped.

0b1 Accesses to MPAM0_EL1 from EL1 are trapped to EL2.

When EL3 is not implemented, this field is reset to 1. When EL3 is implemented, this field resets to UNKNOWN.

TRAPMPAM1EL1, bit [48]

TRAPMPAM1EL1: Trap accesses from EL1 to the MPAM1_EL1 register trap to EL2.

0b0 Accesses to MPAM1_EL1 from EL1 are not trapped.

0b1 Accesses to MPAM1_EL1 from EL1 are trapped to EL2.

When EL3 is not implemented, this field is reset to 1. When EL3 is implemented, this field is resets to UNKNOWN.

PMG_D, bits [47:40]

Performance monitoring group for data accesses.

This field resets to an architecturally UNKNOWN value.

PMG_I, bits [39:32]

Performance monitoring group for instruction accesses.

This field resets to an architecturally UNKNOWN value.

PARTID_D, bits [31:16]

Partition ID for data accesses, including load and store accesses, made from EL2.

This field resets to an architecturally UNKNOWN value.

PARTID_I, bits [15:0]

Partition ID for instruction accesses made from EL2.

This field resets to an architecturally UNKNOWN value.

Accessing the MPAM2_EL2

None of the fields in this register are permitted to be cached in a TLB.

Accesses to this register use the following encodings in the System instruction encoding space:

MRS <Xt>, MPAM2_EL2

op0	op1	CRn	CRm	op2
0b11	0b100	0b1010	0b0101	0b000

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
```

```
        return MPAM2_EL2;
elsif PSTATE.EL == EL3 then
    return MPAM2_EL2;
```

MSR MPAM2_EL2, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b100	0b1010	0b0101	0b000

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        MPAM2_EL2 = X[t];
elsif PSTATE.EL == EL3 then
    MPAM2_EL2 = X[t];
```

MRS <Xt>, MPAM1_EL1

op0	op1	CRn	CRm	op2
0b11	0b000	0b1010	0b0101	0b000

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MPAM2_EL2.TRAPMPAM1EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '111' then
        return NVMem[0x900];
    else
        return MPAM1_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HCR_EL2.E2H == '1' then
        return MPAM2_EL2;
    else
        return MPAM1_EL1;
elsif PSTATE.EL == EL3 then
    return MPAM1_EL1;
```

MSR MPAM1_EL1, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b000	0b1010	0b0101	0b000

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MPAM2_EL2.TRAPMPAM1EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '111' then
        NVMem[0x900] = X[t];
    else
        MPAM1_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HCR_EL2.E2H == '1' then
        MPAM2_EL2 = X[t];
    else
        MPAM1_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    MPAM1_EL1 = X[t];
```

7.4.4 MPAM3_EL3, MPAM3 Register (EL3)

The MPAM3_EL3 characteristics are:

Purpose

Holds information to generate MPAM labels for memory requests when executing at EL3.

Configurations

AArch64 System register MPAM3_EL3[63] is architecturally mapped to AArch64 System register MPAM2_EL2[63] when HaveEL(EL2).

AArch64 System register MPAM3_EL3[63] is architecturally mapped to AArch64 System register MPAM1_EL1[63].

This register is present only when MPAM is implemented. Otherwise, direct accesses to MPAM3_EL3 are UNDEFINED.

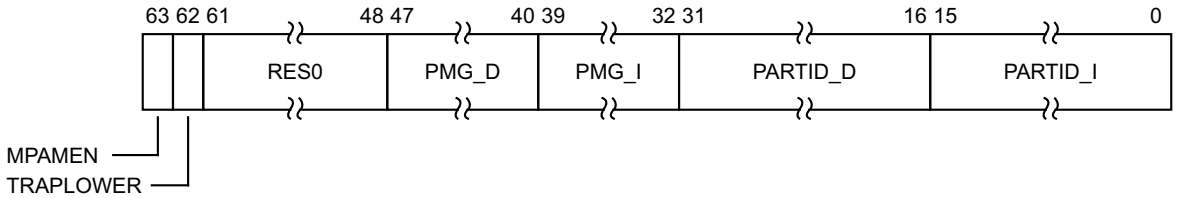
Some or all RW fields of this register have defined reset values. These apply only if the PE resets into an Exception level that is using AArch64. Otherwise, RW fields in this register reset to architecturally UNKNOWN values.

Attributes

MPAM3_EL3 is a 64-bit register.

Field descriptions

The MPAM3_EL3 bit assignments are:



MPAMEN, bit [63]

MPAM Enable: MPAM is enabled when MPAMEN == 1. When disabled, all PARTIDs and PMGs are output as their default value in the corresponding ID space.

Values of this field are:

`0b0` The default PARTID and default PMG are output in MPAM information when executing at any ELn.

`0b1` MPAM information is output based on the MPAMn_ELx register for ELn according the MPAM configuration.

This field is always read/write in MPAM3_EL3.

This field resets to `0`.

TRAPLOWER, bit [62]

Trap direct accesses to any MPAM system registers that are not UNDEFINED from all ELn lower than EL3.

`0b0` Do not force trapping of direct accesses of MPAM system registers to EL3.

`0b1` Force all direct accesses of MPAM system registers to trap to EL3.

On a Cold reset, this field resets to `1`.

Bits [61:48]

Reserved, RES0.

PMG_D, bits [47:40]

Performance monitoring group for data accesses.

This field resets to an architecturally UNKNOWN value.

PMG_I, bits [39:32]

Performance monitoring group for instruction accesses.

PARTID_D, bits [31:16]

Partition ID for data accesses, including load and store accesses, made from EL3.

This field resets to an architecturally UNKNOWN value.

PARTID_I, bits [15:0]

Partition ID for instruction accesses made from EL3.

This field resets to an architecturally UNKNOWN value.

Accessing the MPAM3_EL3

None of the fields in this register are permitted to be cached in a TLB.

Accesses to this register use the following encodings in the System instruction encoding space:

MRS <Xt>, MPAM3_EL3

op0	op1	CRn	CRm	op2
0b11	0b110	0b1010	0b0101	0b000

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    return MPAM3_EL3;
```

MSR MPAM3_EL3, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b110	0b1010	0b0101	0b000

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    MPAM3_EL3 = X[t];
```

7.4.5 MPAMHCR_EL2, MPAM Hypervisor Control Register (EL2)

The MPAMHCR_EL2 characteristics are:

Purpose

MPAMHCR_EL2 controls the PARTID virtualization features of MPAM. It controls the mapping of virtual PARTIDs into physical PARTIDs in MPAM0_EL1 when EL0_VPMEN == 1 and in MPAM1_EL1 when EL1_VPMEN == 1.

Configurations

This register is present only when MPAM is implemented and MPAMIDR_EL1.HAS_HCR == 1. Otherwise, direct accesses to MPAMHCR_EL2 are UNDEFINED.

This register has no effect if EL2 is not enabled in the current Security state.

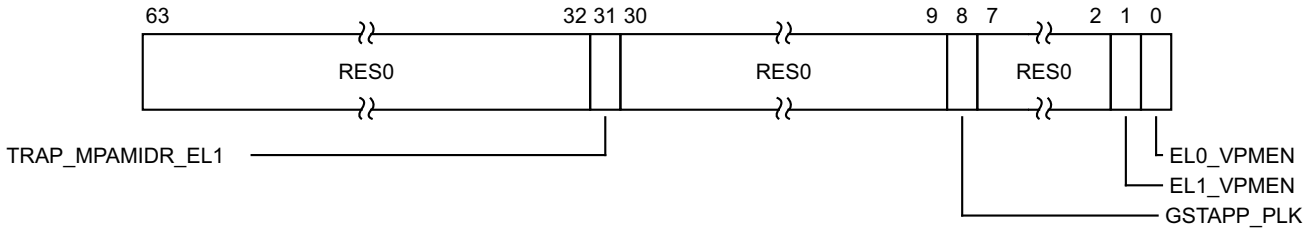
Some or all RW fields of this register have defined reset values. These apply only if the PE resets into an Exception level that is using AArch64. Otherwise, RW fields in this register reset to architecturally UNKNOWN values.

Attributes

MPAMHCR_EL2 is a 64-bit register.

Field descriptions

The MPAMHCR_EL2 bit assignments are:



Bits [63:32]

Reserved, RES0.

TRAP_MPAMIDR_EL1, bit [31]

Trap accesses from EL1 to MPAMIDR_EL1 to EL2.

`0b0` EL1 accesses to MPAMIDR_EL1 return its value and do not trap to EL2.

`0b1` EL1 accesses to MPAMIDR_EL1 trap to EL2.

When EL3 is not implemented, this field is reset to 1. When EL3 is implemented, this field resets to UNKNOWN.

Bits [30:9]

Reserved, RES0.

GSTAPP_PLK, bit [8]

Make the PARTIDs at EL0 the same as the PARTIDs at EL1. When executing at EL0, EL2 is enabled, HCR_EL2.TGE == 0 and GSTAPP_PLK = 1, MPAM1_EL1 is used instead of MPAM0_EL1 to generate MPAM labels for memory requests.

`0b0` MPAM0_EL1 is used to generate MPAM labels when executing at EL0.

`0b1` MPAM1_EL1 is used to generate MPAM labels when executing at EL0 with EL2 enabled and HCR_EL2.TGE == 0. Otherwise MPAM0_EL1 is used.

This field resets to an architecturally UNKNOWN value.

Bits [7:2]

Reserved, RES0.

EL1_VPMEN, bit [1]

Enable the virtual PARTID mapping of the PARTID fields in MPAM1_EL1 when executing at EL1. This bit also enables virtual PARTID mapping when MPAM1_EL1 is used to generate MPAM labels for memory requests at EL0 due to GSTAPP_PLK == 1.

0b0 MPAM1_EL1.PARTID_I and MPAM1_EL1.PARTID_D are physical PARTIDs that are used to label memory system requests.

0b1 MPAM1_EL1.PARTID_I and MPAM1_EL1.PARTID_D are virtual PARTIDs that are used to index the PhyPARTID fields of MPAMVPM0_EL2 to MPAMVPM7_EL2 registers to map the virtual PARTID into a physical PARTID to label memory system requests.

This field resets to an architecturally UNKNOWN value.

EL0_VPMEN, bit [0]

Enable the virtual PARTID mapping of the PARTID fields of MPAM0_EL1 unless HCR_EL2.E2H == '1' and HCR_EL2.TGE == '1'.

When HCR_EL2.E2H == '1' and HCR_EL2.TGE == '1', EL0_VPMEN is ignored and MPAM0_EL1 PARTID fields are not mapped.

When MPAMHCR_EL2.GSTAPP_PLK == '1' and HCR_EL2.TGE == '0', MPAM1_EL1 is used as the source of PARTIDs and the virtual PARTID mapping of MPAM1_EL1 PARTIDs is controlled by MPAMHCR_EL2.EL1_VPMEN.

0b0 MPAM0_EL1.PARTID_I and MPAM0_EL1.PARTID_D are physical PARTIDs that are used to label memory system requests.

0b1 MPAM0_EL1.PARTID_I and MPAM0_EL1.PARTID_D are virtual PARTIDs that are used to index the PhyPARTID fields of MPAMVPM0_EL2 to MPAMVPM7_EL2 registers to map the virtual PARTID into a physical PARTID to label memory system requests.

This field resets to an architecturally UNKNOWN value.

Accessing the MPAMHCR_EL2

Accesses to this register use the following encodings in the System instruction encoding space:

MRS <Xt>, MPAMHCR_EL2

op0	op1	CRn	CRm	op2
0b11	0b100	0b1010	0b0100	0b000

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        return NVMem[0x930];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
```

```
        return MPAMHCR_EL2;
elsif PSTATE.EL == EL3 then
    return MPAMHCR_EL2;
```

MSR MPAMHCR_EL2, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b100	0b1010	0b0100	0b000

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        NVMem[0x930] = X[t];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        MPAMHCR_EL2 = X[t];
elsif PSTATE.EL == EL3 then
    MPAMHCR_EL2 = X[t];
```

7.4.6 MPAMIDR_EL1, MPAM ID Register (EL1)

The MPAMIDR_EL1 characteristics are:

Purpose

MPAMIDR_EL1 indicates the presence and maximum PARTID and PMG values supported in the implementation. It also indicates whether the implementation supports MPAM virtualization.

Configurations

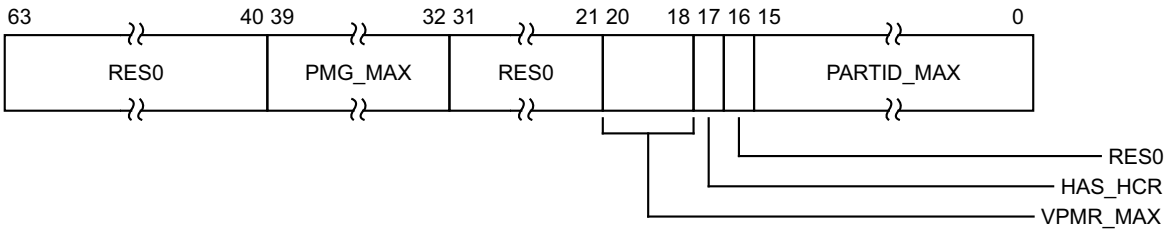
This register is present only when MPAM is implemented. Otherwise, direct accesses to MPAMIDR_EL1 are UNDEFINED.

Attributes

MPAMIDR_EL1 is a 64-bit register.

Field descriptions

The MPAMIDR_EL1 bit assignments are:



MPAMIDR_EL1 indicates the MPAM implementation parameters of the PE.

Bits [63:40]

Reserved, RES0.

PMG_MAX, bits [39:32]

The largest value of PMG that the implementation can generate. The PMG_I and PMG_D fields of every MPAMn_ELx must implement at least enough bits to represent PMG_MAX.

Bits [31:21]

Reserved, RES0.

VPMR_MAX, bits [20:18]

If HAS_HCR == 0, VPMR_MAX must be `0b000`. Otherwise, it indicates the maximum register index n for the MPAMVPM<n>_EL2 registers.

HAS_HCR, bit [17]

HAS_HCR indicates that the PE implementation supports MPAM virtualization, including MPAMHCR_EL2, MPAMVPMV_EL2 and MPAMVPM<n>_EL2 with n in the range 0 to VPMR_MAX. Must be 0 if EL2 is not implemented in either security state.

`0b0` MPAM virtualization is not supported.

`0b1` MPAM virtualization is supported.

Bit [16]

Reserved, RES0.

PARTID_MAX, bits [15:0]

The largest value of PMG that the implementation can generate. The PARTED_I and PARTID_D fields of every MPAMn_ELx must implement at least enough bits to represent PARTID_MAX.

Accessing the MPAMIDR_EL1

Accesses to this register use the following encodings in the System instruction encoding space:

MRS <Xt>, MPAMIDR_EL1

op0	op1	CRn	CRm	op2
0b11	0b000	0b1010	0b0100	0b100

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MPAMIDR_EL1.HAS_HCR == '1' &&
MPAMHCR_EL2.TRAP_MPAMIDR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        return MPAMIDR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return MPAMIDR_EL1;
elsif PSTATE.EL == EL3 then
    return MPAMIDR_EL1;
```

7.4.7 MPAMVPM0_EL2, MPAM Virtual PARTID Mapping Register 0

The MPAMVPM0_EL2 characteristics are:

Purpose

MPAMVPM0_EL2 provides mappings from virtual PARTIDs 0 - 3 to physical PARTIDs.

MPAMIDR_EL1.VPMR_MAX field gives the index of the highest implemented MPAMVPM<n>_EL2 register. VPMR_MAX can be as large as 7 (8 registers) or 32 virtual PARTIDs. If MPAMIDR_EL1.VPMR_MAX == 0, there is only a single MPAMVPM<n>_EL2 register, MPAMVPM0_EL2.

Virtual PARTID mapping is enabled by MPAMHCR_EL2.EL1_VPMEN for PARTIDs in MPAM1_EL1 and by MPAMHCR_EL2.EL0_VPMEN for PARTIDs in MPAM0_EL1.

A virtual-to-physical PARTID mapping entry, PhyPARTID<n>, is only valid when the MPAMVPMV_EL2.VPM_V bit in bit position n is set to 1.

Configurations

This register is present only when MPAM is implemented and MPAMIDR_EL1.HAS_HCR == 1. Otherwise, direct accesses to MPAMVPM0_EL2 are UNDEFINED.

This register has no effect if EL2 is not enabled in the current Security state.

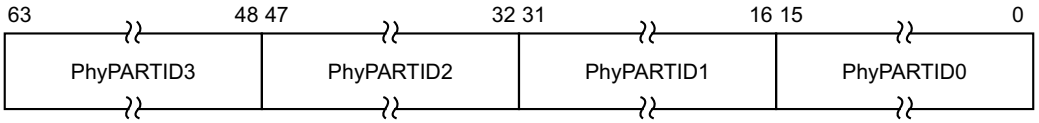
RW fields in this register reset to architecturally UNKNOWN values.

Attributes

MPAMVPM0_EL2 is a 64-bit register.

Field descriptions

The MPAMVPM0_EL2 bit assignments are:



PhyPARTID3, bits [63:48]

Virtual PARTID Mapping Entry for virtual PARTID 3. PhyPARTID3 gives the mapping of virtual PARTID 3 to a physical PARTID.

This field resets to an architecturally UNKNOWN value.

PhyPARTID2, bits [47:32]

Virtual PARTID Mapping Entry for virtual PARTID 2. PhyPARTID2 gives the mapping of virtual PARTID 2 to a physical PARTID.

This field resets to an architecturally UNKNOWN value.

PhyPARTID1, bits [31:16]

Virtual PARTID Mapping Entry for virtual PARTID 1. PhyPARTID1 gives the mapping of virtual PARTID 1 to a physical PARTID.

This field resets to an architecturally UNKNOWN value.

PhyPARTID0, bits [15:0]

Virtual PARTID Mapping Entry for virtual PARTID 0. PhyPARTID0 gives the mapping of virtual PARTID 0 to a physical PARTID.

This field resets to an architecturally UNKNOWN value.

Accessing the MPAMVPM0_EL2

Accesses to this register use the following encodings in the System instruction encoding space:

MRS <Xt>, MPAMVPM0_EL2

op0	op1	CRn	CRm	op2
0b11	0b100	0b1010	0b0110	0b000

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        return NVMem[0x940];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return MPAMVPM0_EL2;
elsif PSTATE.EL == EL3 then
    return MPAMVPM0_EL2;
```

MSR MPAMVPM0_EL2, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b100	0b1010	0b0110	0b000

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        NVMem[0x940] = X[t];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        MPAMVPM0_EL2 = X[t];
elsif PSTATE.EL == EL3 then
    MPAMVPM0_EL2 = X[t];
```

7.4.8 MPAMVPM1_EL2, MPAM Virtual PARTID Mapping Register 1

The MPAMVPM1_EL2 characteristics are:

Purpose

MPAMVPM1_EL2 provides mappings from virtual PARTIDs 4 - 7 to physical PARTIDs.

MPAMIDR_EL1.VPMR_MAX field gives the index of the highest implemented MPAMVPM0_EL2 to MPAMVPM7_EL2 registers. VPMR_MAX can be as large as 7 (8 registers) or 32 virtual PARTIDs. If MPAMIDR_EL1.VPMR_MAX == 0, there is only a single MPAMVPM<n>_EL2 register, MPAMVPM0_EL2.

Virtual PARTID mapping is enabled by MPAMHCR_EL2.EL1_VPMEN for PARTIDs in MPAM1_EL1 and by MPAMHCR_EL2.EL0_VPMEN for PARTIDs in MPAM0_EL1.

A virtual-to-physical PARTID mapping entry, PhyPARTID<n>, is only valid when the MPAMVPMV_EL2.VPM_V bit in bit position n is set to 1.

Configurations

This register is present only when MPAM is implemented, MPAMIDR_EL1.HAS_HCR == 1 and MPAMIDR_EL1.VPMR_MAX > 0. Otherwise, direct accesses to MPAMVPM1_EL2 are UNDEFINED.

This register has no effect if EL2 is not enabled in the current Security state.

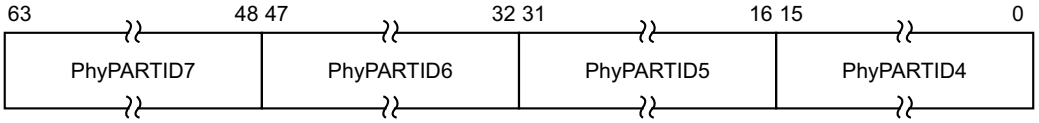
RW fields in this register reset to architecturally UNKNOWN values.

Attributes

MPAMVPM1_EL2 is a 64-bit register.

Field descriptions

The MPAMVPM1_EL2 bit assignments are:



PhyPARTID7, bits [63:48]

Virtual PARTID Mapping Entry for virtual PARTID 7. PhyPARTID7 gives the mapping of virtual PARTID 7 to a physical PARTID.

This field resets to an architecturally UNKNOWN value.

PhyPARTID6, bits [47:32]

Virtual PARTID Mapping Entry for virtual PARTID 6. PhyPARTID6 gives the mapping of virtual PARTID 6 to a physical PARTID.

This field resets to an architecturally UNKNOWN value.

PhyPARTID5, bits [31:16]

Virtual PARTID Mapping Entry for virtual PARTID 5. PhyPARTID5 gives the mapping of virtual PARTID 5 to a physical PARTID.

This field resets to an architecturally UNKNOWN value.

PhyPARTID4, bits [15:0]

Virtual PARTID Mapping Entry for virtual PARTID 4. PhyPARTID4 gives the mapping of virtual PARTID 4 to a physical PARTID.

This field resets to an architecturally UNKNOWN value.

Accessing the MPAMVPM1_EL2

Accesses to this register use the following encodings in the System instruction encoding space:

MRS <Xt>, MPAMVPM1_EL2

op0	op1	CRn	CRm	op2
0b11	0b100	0b1010	0b0110	0b001

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        return NVMem[0x948];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return MPAMVPM1_EL2;
elsif PSTATE.EL == EL3 then
    return MPAMVPM1_EL2;
```

MSR MPAMVPM1_EL2, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b100	0b1010	0b0110	0b001

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        NVMem[0x948] = X[t];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        MPAMVPM1_EL2 = X[t];
elsif PSTATE.EL == EL3 then
    MPAMVPM1_EL2 = X[t];
```

7.4.9 MPAMVPM2_EL2, MPAM Virtual PARTID Mapping Register 2

The MPAMVPM2_EL2 characteristics are:

Purpose

MPAMVPM2_EL2 provides mappings from virtual PARTIDs 8 - 11 to physical PARTIDs.

MPAMIDR_EL1.VPMR_MAX field gives the index of the highest implemented MPAMVPM0_EL2 to MPAMVPM7_EL2 registers. VPMR_MAX can be as large as 7 (8 registers) or 32 virtual PARTIDs. If MPAMIDR_EL1.VPMR_MAX == 0, there is only a single MPAMVPM<n>_EL2 register, MPAMVPM0_EL2.

Virtual PARTID mapping is enabled by MPAMHCR_EL2.EL1_VPMEN for PARTIDs in MPAM1_EL1 and by MPAMHCR_EL2.EL0_VPMEN for PARTIDs in MPAM0_EL1.

A virtual-to-physical PARTID mapping entry, PhyPARTID<n>, is only valid when the MPAMVPMV_EL2.VPM_V bit in bit position n is set to 1.

Configurations

This register is present only when MPAM is implemented, MPAMIDR_EL1.HAS_HCR == 1 and MPAMIDR_EL1.VPMR_MAX > 1. Otherwise, direct accesses to MPAMVPM2_EL2 are UNDEFINED.

This register has no effect if EL2 is not enabled in the current Security state.

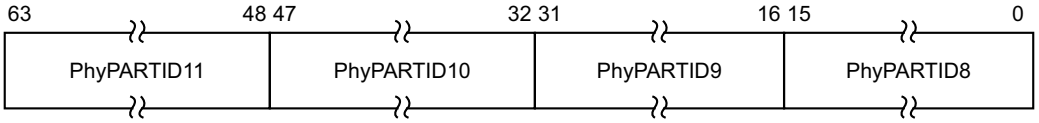
RW fields in this register reset to architecturally UNKNOWN values.

Attributes

MPAMVPM2_EL2 is a 64-bit register.

Field descriptions

The MPAMVPM2_EL2 bit assignments are:



PhyPARTID11, bits [63:48]

Virtual PARTID Mapping Entry for virtual PARTID 11. PhyPARTID11 gives the mapping of virtual PARTID 11 to a physical PARTID.

This field resets to an architecturally UNKNOWN value.

PhyPARTID10, bits [47:32]

Virtual PARTID Mapping Entry for virtual PARTID 10. PhyPARTID10 gives the mapping of virtual PARTID 10 to a physical PARTID.

This field resets to an architecturally UNKNOWN value.

PhyPARTID9, bits [31:16]

Virtual PARTID Mapping Entry for virtual PARTID 9. PhyPARTID9 gives the mapping of virtual PARTID 9 to a physical PARTID.

This field resets to an architecturally UNKNOWN value.

PhyPARTID8, bits [15:0]

Virtual PARTID Mapping Entry for virtual PARTID 8. PhyPARTID8 gives the mapping of virtual PARTID 8 to a physical PARTID.

This field resets to an architecturally UNKNOWN value.

Accessing the MPAMVPM2_EL2

Accesses to this register use the following encodings in the System instruction encoding space:

MRS <Xt>, MPAMVPM2_EL2

op0	op1	CRn	CRm	op2
0b11	0b100	0b1010	0b0110	0b010

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        return NVMem[0x950];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return MPAMVPM2_EL2;
elsif PSTATE.EL == EL3 then
    return MPAMVPM2_EL2;
```

MSR MPAMVPM2_EL2, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b100	0b1010	0b0110	0b010

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        NVMem[0x950] = X[t];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        MPAMVPM2_EL2 = X[t];
elsif PSTATE.EL == EL3 then
    MPAMVPM2_EL2 = X[t];
```

7.4.10 MPAMVPM3_EL2, MPAM Virtual PARTID Mapping Register 3

The MPAMVPM3_EL2 characteristics are:

Purpose

MPAMVPM3_EL2 provides mappings from virtual PARTIDs 12 - 15 to physical PARTIDs.

MPAMIDR_EL1.VPMR_MAX field gives the index of the highest implemented MPAMVPM<n>_EL2 registers. VPMR_MAX can be as large as 7 (8 registers) or 32 virtual PARTIDs. If MPAMIDR_EL1.VPMR_MAX == 0, there is only a single MPAMVPM<n>_EL2 register, MPAMVPM0_EL2.

Virtual PARTID mapping is enabled by MPAMHCR_EL2.EL1_VPMEN for PARTIDs in MPAM1_EL1 and by MPAMHCR_EL2.EL0_VPMEN for PARTIDs in MPAM0_EL1.

A virtual-to-physical PARTID mapping entry, PhyPARTID<n>, is only valid when the MPAMVPMV_EL2.VPM_V bit in bit position n is set to 1.

Configurations

This register is present only when MPAM is implemented, MPAMIDR_EL1.HAS_HCR == 1 and MPAMIDR_EL1.VPMR_MAX > 2. Otherwise, direct accesses to MPAMVPM3_EL2 are UNDEFINED.

This register has no effect if EL2 is not enabled in the current Security state.

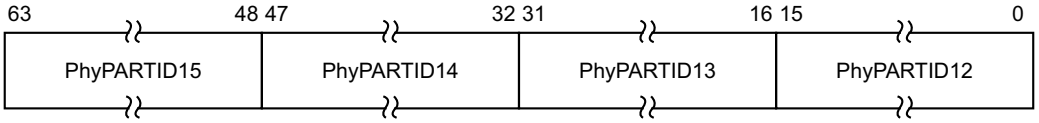
RW fields in this register reset to architecturally UNKNOWN values.

Attributes

MPAMVPM3_EL2 is a 64-bit register.

Field descriptions

The MPAMVPM3_EL2 bit assignments are:



PhyPARTID15, bits [63:48]

Virtual PARTID Mapping Entry for virtual PARTID 15. PhyPARTID15 gives the mapping of virtual PARTID 15 to a physical PARTID.

This field resets to an architecturally UNKNOWN value.

PhyPARTID14, bits [47:32]

Virtual PARTID Mapping Entry for virtual PARTID 14. PhyPARTID14 gives the mapping of virtual PARTID 14 to a physical PARTID.

This field resets to an architecturally UNKNOWN value.

PhyPARTID13, bits [31:16]

Virtual PARTID Mapping Entry for virtual PARTID 13. PhyPARTID13 gives the mapping of virtual PARTID 13 to a physical PARTID.

This field resets to an architecturally UNKNOWN value.

PhyPARTID12, bits [15:0]

Virtual PARTID Mapping Entry for virtual PARTID 12. PhyPARTID12 gives the mapping of virtual PARTID 12 to a physical PARTID.

This field resets to an architecturally UNKNOWN value.

Accessing the MPAMVPM3_EL2

Accesses to this register use the following encodings in the System instruction encoding space:

MRS <Xt>, MPAMVPM3_EL2

op0	op1	CRn	CRm	op2
0b11	0b100	0b1010	0b0110	0b011

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        return NVMem[0x958];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return MPAMVPM3_EL2;
elsif PSTATE.EL == EL3 then
    return MPAMVPM3_EL2;
```

MSR MPAMVPM3_EL2, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b100	0b1010	0b0110	0b011

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        NVMem[0x958] = X[t];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        MPAMVPM3_EL2 = X[t];
elsif PSTATE.EL == EL3 then
    MPAMVPM3_EL2 = X[t];
```

7.4.11 MPAMVPM4_EL2, MPAM Virtual PARTID Mapping Register 4

The MPAMVPM4_EL2 characteristics are:

Purpose

MPAMVPM4_EL2 provides mappings from virtual PARTIDs 16 - 19 to physical PARTIDs.

MPAMIDR_EL1.VPMR_MAX field gives the index of the highest implemented MPAMVPM<n>_EL2 registers. VPMR_MAX can be as large as 7 (8 registers) or 32 virtual PARTIDs. If MPAMIDR_EL1.VPMR_MAX == 0, there is only a single MPAMVPM<n>_EL2 register, MPAMVPM0_EL2.

Virtual PARTID mapping is enabled by MPAMHCR_EL2.EL1_VPMEN for PARTIDs in MPAM1_EL1 and by MPAMHCR_EL2.EL0_VPMEN for PARTIDs in MPAM0_EL1.

A virtual-to-physical PARTID mapping entry, PhyPARTID<n>, is only valid when the MPAMVPMV_EL2.VPM_V bit in bit position n is set to 1.

Configurations

This register is present only when MPAM is implemented, MPAMIDR_EL1.HAS_HCR == 1 and MPAMIDR_EL1.VPMR_MAX > 3. Otherwise, direct accesses to MPAMVPM4_EL2 are UNDEFINED.

This register has no effect if EL2 is not enabled in the current Security state.

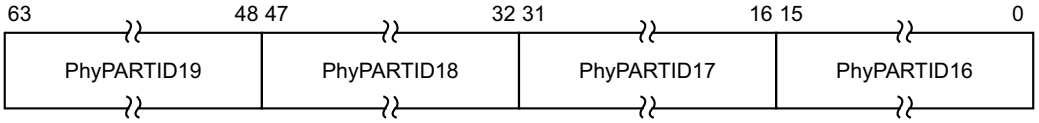
RW fields in this register reset to architecturally UNKNOWN values.

Attributes

MPAMVPM4_EL2 is a 64-bit register.

Field descriptions

The MPAMVPM4_EL2 bit assignments are:



PhyPARTID19, bits [63:48]

Virtual PARTID Mapping Entry for virtual PARTID 19. PhyPARTID19 gives the mapping of virtual PARTID 19 to a physical PARTID.

This field resets to an architecturally UNKNOWN value.

PhyPARTID18, bits [47:32]

Virtual PARTID Mapping Entry for virtual PARTID 18. PhyPARTID18 gives the mapping of virtual PARTID 18 to a physical PARTID.

This field resets to an architecturally UNKNOWN value.

PhyPARTID17, bits [31:16]

Virtual PARTID Mapping Entry for virtual PARTID 17. PhyPARTID17 gives the mapping of virtual PARTID 17 to a physical PARTID.

This field resets to an architecturally UNKNOWN value.

PhyPARTID16, bits [15:0]

Virtual PARTID Mapping Entry for virtual PARTID 16. PhyPARTID16 gives the mapping of virtual PARTID 16 to a physical PARTID.

This field resets to an architecturally UNKNOWN value.

Accessing the MPAMVPM4_EL2

Accesses to this register use the following encodings in the System instruction encoding space:

MRS <Xt>, MPAMVPM4_EL2

op0	op1	CRn	CRm	op2
0b11	0b100	0b1010	0b0110	0b100

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        return NVMem[0x960];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return MPAMVPM4_EL2;
elsif PSTATE.EL == EL3 then
    return MPAMVPM4_EL2;
```

MSR MPAMVPM4_EL2, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b100	0b1010	0b0110	0b100

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        NVMem[0x960] = X[t];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        MPAMVPM4_EL2 = X[t];
elsif PSTATE.EL == EL3 then
    MPAMVPM4_EL2 = X[t];
```

7.4.12 MPAMVPM5_EL2, MPAM Virtual PARTID Mapping Register 5

The MPAMVPM5_EL2 characteristics are:

Purpose

MPAMVPM5_EL2 provides mappings from virtual PARTIDs 20 - 23 to physical PARTIDs.

MPAMIDR_EL1.VPMR_MAX field gives the index of the highest implemented MPAMVPM<n>_EL2 registers. VPMR_MAX can be as large as 7 (8 registers) or 32 virtual PARTIDs. If MPAMIDR_EL1.VPMR_MAX == 0, there is only a single MPAMVPM<n>_EL2 register, MPAMVPM0_EL2.

Virtual PARTID mapping is enabled by MPAMHCR_EL2.EL1_VPMEN for PARTIDs in MPAM1_EL1 and by MPAMHCR_EL2.EL0_VPMEN for PARTIDs in MPAM0_EL1.

A virtual-to-physical PARTID mapping entry, PhyPARTID<n>, is only valid when the MPAMVPMV_EL2.VPM_V bit in bit position n is set to 1.

Configurations

This register is present only when MPAM is implemented, MPAMIDR_EL1.HAS_HCR == 1 and MPAMIDR_EL1.VPMR_MAX > 4. Otherwise, direct accesses to MPAMVPM5_EL2 are UNDEFINED.

This register has no effect if EL2 is not enabled in the current Security state.

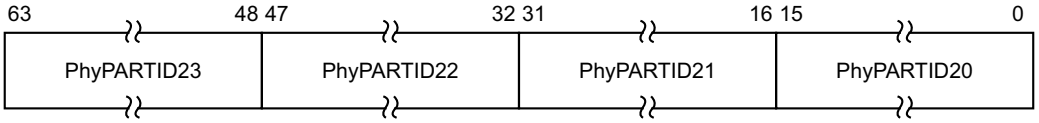
RW fields in this register reset to architecturally UNKNOWN values.

Attributes

MPAMVPM5_EL2 is a 64-bit register.

Field descriptions

The MPAMVPM5_EL2 bit assignments are:



PhyPARTID23, bits [63:48]

Virtual PARTID Mapping Entry for virtual PARTID 23. PhyPARTID23 gives the mapping of virtual PARTID 23 to a physical PARTID.

This field resets to an architecturally UNKNOWN value.

PhyPARTID22, bits [47:32]

Virtual PARTID Mapping Entry for virtual PARTID 22. PhyPARTID22 gives the mapping of virtual PARTID 22 to a physical PARTID.

This field resets to an architecturally UNKNOWN value.

PhyPARTID21, bits [31:16]

Virtual PARTID Mapping Entry for virtual PARTID 21. PhyPARTID21 gives the mapping of virtual PARTID 21 to a physical PARTID.

This field resets to an architecturally UNKNOWN value.

PhyPARTID20, bits [15:0]

Virtual PARTID Mapping Entry for virtual PARTID 20. PhyPARTID20 gives the mapping of virtual PARTID 20 to a physical PARTID.

This field resets to an architecturally UNKNOWN value.

Accessing the MPAMVPM5_EL2

Accesses to this register use the following encodings in the System instruction encoding space:

MRS <Xt>, MPAMVPM5_EL2

op0	op1	CRn	CRm	op2
0b11	0b100	0b1010	0b0110	0b101

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        return NVMem[0x968];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return MPAMVPM5_EL2;
elsif PSTATE.EL == EL3 then
    return MPAMVPM5_EL2;
```

MSR MPAMVPM5_EL2, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b100	0b1010	0b0110	0b101

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        NVMem[0x968] = X[t];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        MPAMVPM5_EL2 = X[t];
elsif PSTATE.EL == EL3 then
    MPAMVPM5_EL2 = X[t];
```

7.4.13 MPAMVPM6_EL2, MPAM Virtual PARTID Mapping Register 6

The MPAMVPM6_EL2 characteristics are:

Purpose

MPAMVPM6_EL2 provides mappings from virtual PARTIDs 24 - 27 to physical PARTIDs.

MPAMIDR_EL1.VPMR_MAX field gives the index of the highest implemented MPAMVPM<n>_EL2 registers. VPMR_MAX can be as large as 7 (8 registers) or 32 virtual PARTIDs. If MPAMIDR_EL1.VPMR_MAX == 0, there is only a single MPAMVPM<n>_EL2 register, MPAMVPM0_EL2.

Virtual PARTID mapping is enabled by MPAMHCR_EL2.EL1_VPMEN for PARTIDs in MPAM1_EL1 and by MPAMHCR_EL2.EL0_VPMEN for PARTIDs in MPAM0_EL1.

A virtual-to-physical PARTID mapping entry, PhyPARTID<n>, is only valid when the MPAMVPMV_EL2.VPM_V bit in bit position n is set to 1.

Configurations

This register is present only when MPAM is implemented, MPAMIDR_EL1.HAS_HCR == 1 and MPAMIDR_EL1.VPMR_MAX > 5. Otherwise, direct accesses to MPAMVPM6_EL2 are UNDEFINED.

This register has no effect if EL2 is not enabled in the current Security state.

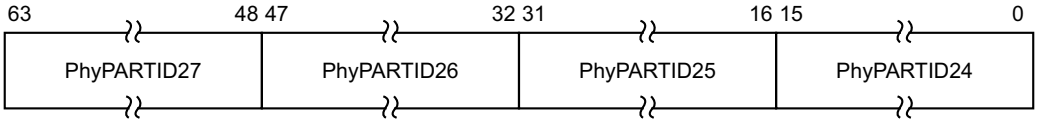
RW fields in this register reset to architecturally UNKNOWN values.

Attributes

MPAMVPM6_EL2 is a 64-bit register.

Field descriptions

The MPAMVPM6_EL2 bit assignments are:



PhyPARTID27, bits [63:48]

Virtual PARTID Mapping Entry for virtual PARTID 27. PhyPARTID27 gives the mapping of virtual PARTID 27 to a physical PARTID.

This field resets to an architecturally UNKNOWN value.

PhyPARTID26, bits [47:32]

Virtual PARTID Mapping Entry for virtual PARTID 26. PhyPARTID26 gives the mapping of virtual PARTID 26 to a physical PARTID.

This field resets to an architecturally UNKNOWN value.

PhyPARTID25, bits [31:16]

Virtual PARTID Mapping Entry for virtual PARTID 25. PhyPARTID25 gives the mapping of virtual PARTID 25 to a physical PARTID.

This field resets to an architecturally UNKNOWN value.

PhyPARTID24, bits [15:0]

Virtual PARTID Mapping Entry for virtual PARTID 24. PhyPARTID24 gives the mapping of virtual PARTID 24 to a physical PARTID.

This field resets to an architecturally UNKNOWN value.

Accessing the MPAMVPM6_EL2

Accesses to this register use the following encodings in the System instruction encoding space:

MRS <Xt>, MPAMVPM6_EL2

op0	op1	CRn	CRm	op2
0b11	0b100	0b1010	0b0110	0b110

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        return NVMem[0x970];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return MPAMVPM6_EL2;
elsif PSTATE.EL == EL3 then
    return MPAMVPM6_EL2;
```

MSR MPAMVPM6_EL2, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b100	0b1010	0b0110	0b110

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        NVMem[0x970] = X[t];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        MPAMVPM6_EL2 = X[t];
elsif PSTATE.EL == EL3 then
    MPAMVPM6_EL2 = X[t];
```

7.4.14 MPAMVPM7_EL2, MPAM Virtual PARTID Mapping Register 7

The MPAMVPM7_EL2 characteristics are:

Purpose

MPAMVPM7_EL2 provides mappings from virtual PARTIDs 28 - 31 to physical PARTIDs.

MPAMIDR_EL1.VPMR_MAX field gives the index of the highest implemented MPAMVPM<n>_EL2 registers. VPMR_MAX can be as large as 7 (8 registers) or 32 virtual PARTIDs. If MPAMIDR_EL1.VPMR_MAX == 0, there is only a single MPAMVPM<n>_EL2 register, MPAMVPM0_EL2.

Virtual PARTID mapping is enabled by MPAMHCR_EL2.EL1_VPMEN for PARTIDs in MPAM1_EL1 and by MPAMHCR_EL2.EL0_VPMEN for MPAM0_EL1.

A virtual-to-physical PARTID mapping entry, PhyPARTID<n>, is only valid when the MPAMVPMV_EL2.VPM_V bit in bit position n is set to 1.

Configurations

This register is present only when MPAM is implemented, MPAMIDR_EL1.HAS_HCR == 1 and MPAMIDR_EL1.VPMR_MAX == 7. Otherwise, direct accesses to MPAMVPM7_EL2 are UNDEFINED.

This register has no effect if EL2 is not enabled in the current Security state.

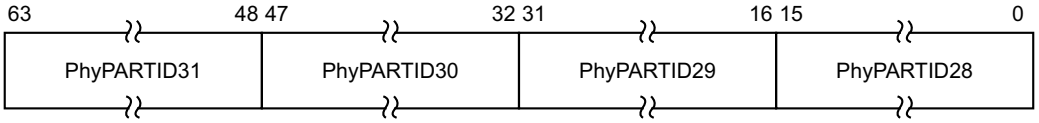
RW fields in this register reset to architecturally UNKNOWN values.

Attributes

MPAMVPM7_EL2 is a 64-bit register.

Field descriptions

The MPAMVPM7_EL2 bit assignments are:



PhyPARTID31, bits [63:48]

Virtual PARTID Mapping Entry for virtual PARTID 31. PhyPARTID31 gives the mapping of virtual PARTID 31 to a physical PARTID.

This field resets to an architecturally UNKNOWN value.

PhyPARTID30, bits [47:32]

Virtual PARTID Mapping Entry for virtual PARTID 30. PhyPARTID30 gives the mapping of virtual PARTID 30 to a physical PARTID.

This field resets to an architecturally UNKNOWN value.

PhyPARTID29, bits [31:16]

Virtual PARTID Mapping Entry for virtual PARTID 29. PhyPARTID29 gives the mapping of virtual PARTID 29 to a physical PARTID.

This field resets to an architecturally UNKNOWN value.

PhyPARTID28, bits [15:0]

Virtual PARTID Mapping Entry for virtual PARTID 28. PhyPARTID28 gives the mapping of virtual PARTID 28 to a physical PARTID.

This field resets to an architecturally UNKNOWN value.

Accessing the MPAMVPM7_EL2

Accesses to this register use the following encodings in the System instruction encoding space:

MRS <Xt>, MPAMVPM7_EL2

op0	op1	CRn	CRm	op2
0b11	0b100	0b1010	0b0110	0b111

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        return NVMem[0x978];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return MPAMVPM7_EL2;
elsif PSTATE.EL == EL3 then
    return MPAMVPM7_EL2;
```

MSR MPAMVPM7_EL2, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b100	0b1010	0b0110	0b111

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        NVMem[0x978] = X[t];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        MPAMVPM7_EL2 = X[t];
elsif PSTATE.EL == EL3 then
    MPAMVPM7_EL2 = X[t];
```

7.4.15 MPAMVPMV_EL2, MPAM Virtual Partition Mapping Valid Register

The MPAMVPMV_EL2 characteristics are:

Purpose

Valid bits for virtual PARTID mapping entries. Each bit m corresponds to virtual PARTID mapping entry m in the MPAMVPM<n>_EL2 registers where n = m >> 2.

Configurations

This register is present only when MPAM is implemented and MPAMIDR_EL1.HAS_HCR == 1. Otherwise, direct accesses to MPAMVPMV_EL2 are UNDEFINED.

This register has no effect if EL2 is not enabled in the current Security state.

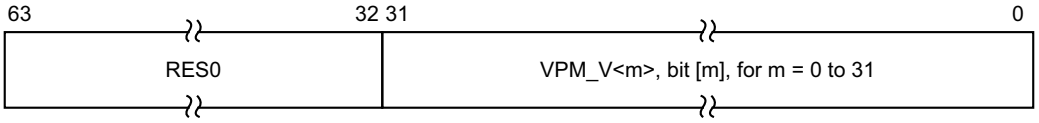
RW fields in this register reset to architecturally UNKNOWN values.

Attributes

MPAMVPMV_EL2 is a 64-bit register.

Field descriptions

The MPAMVPMV_EL2 bit assignments are:



Bits [63:32]

Reserved, RES0.

VPM_V<m>, bit [m], for m = 0 to 31

Contains valid bit for virtual PARTID mapping entry corresponding to virtual PARTID<m>.

This field resets to an architecturally UNKNOWN value.

Accessing the MPAMVPMV_EL2

Accesses to this register use the following encodings in the System instruction encoding space:

MRS <Xt>, MPAMVPMV_EL2

op0	op1	CRn	CRm	op2
0b11	0b100	0b1010	0b0100	0b001

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        return NVMem[0x938];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    return MPAMVPMV_EL2;
elsif PSTATE.EL == EL3 then
    return MPAMVPMV_EL2;
```

MSR MPAMVPMV_EL2, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b100	0b1010	0b0100	0b001

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        NVMem[0x938] = X[t];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    MPAMVPMV_EL2 = X[t];
elsif PSTATE.EL == EL3 then
    MPAMVPMV_EL2 = X[t];
```

7.5 MPAM enable

A single, writable MPAMEN bit exists only in the MPAMn_ELx register for the highest implemented ELn. The highest EL might be EL3, EL2, or EL1. For example, if the highest implemented level is EL3, MPAM3_EL3 would contain the MPAMEN bit. A read-only copy of MPAMEN is present in each of MPAM2_EL2 and MPAM1_EL1 that is implemented and not the highest implemented EL.

When the MPAMEN bit is set, MPAM PARTID and PMG are generated as described in this document. When the MPAMEN bit is clear, default values are generated for MPAM physical PARTID and PMG with MPAM_NS reflecting the PE's current security state. See *PARTID spaces and properties* on page 3-30 for more on default IDs.

The MPAMEN bit is reset to 0.

7.6 Lower-EL MPAM register access trapping

When MPAM3_EL3.TRAPLOWER == 1, direct accesses to MPAM system registers from EL1 or EL2 that are not UNDEFINED trap to EL3. These registers remain accessible from EL3, thus allowing EL3 to set up the MPAM environments for lower levels that are not MPAM-aware.

MPAM3_EL3.TRAPLOWER traps have priority over traps controlled by MPAM2_EL2 and MPAMHCR_EL2.

HCR_EL2.NV == 1 alters the behavior of TRAPLOWER because it makes some _EL2 and _EL12 registers that would be UNDEFINED at EL1 trap to EL2. NV == 1 does not affect accesses from EL0, EL2, or EL3. So with NV == 1 and TRAPLOWER == 1, accesses to MPAM registers from EL2 are trapped to EL3. See *Nested virtualization extension* on page 6-67 for details.

HCR_EL2.NV2 == 1 alters the behavior of TRAPLOWER because it converts accesses to some _EL2 and EL12 registers from EL1 that would be undefined into accesses to memory. See *Enhanced nested virtualization extension* on page 6-67 for details.

7.7 Reset

MPAM system registers are only minimally reset.

- The MPAMEN bit must be set to 0 by warm or cold reset of the PE.
- The MPAM3_EL3.TRAPLOWER bit must be set to 1 by warm or cold reset of the PE.

The MPAM2_EL2.TRAPMPAM1EL1, MPAM2_EL2.TRAPMPAM0EL1, and MPAMHCR_EL2.TRAP_MPAMIDR_EL1 bits are not reset if EL3 exists, but all three bits are reset to 1 if EL3 does not exist.

7.8 Unimplemented exception levels

The Armv8-A architecture permits implementations with or without EL3. Independent from the choice of whether EL3 is implemented or implemented but disabled, the architecture permits implementations with or without EL2.

Even if Non-secure EL2 is implemented, Secure EL2 does not exist in the Arm v8 Architecture before v8.4. Secure EL2 is permitted to be implemented or not implemented in a v8.4 or later implementations. If Secure EL2 is implemented, it may be enabled or disabled by SCR_EL3.EEL2.

EL1 and EL0 are required in all implementations.

Generally, control bits in the MPAMn_ELx registers and MPAMHCR_EL2 for an unimplemented exception level are treated as inactive by all other MPAM exception levels. Details are given in the following subsections.

7.8.1 Effects if EL3 is not implemented

- MPAM3_EL3 is UNDEFINED.
- MPAM3_EL3.TRAPLOWER: All references to this bit behave as if it == 0.
- MPAM2_EL2.MPAMEN is present and RW if EL2 exists. If EL2 does not exist, MPAM1_EL1.MPAMEN is present and RW.

7.8.2 Effects if EL2 is implemented in neither security state

- MPAM2_EL2 is RES0 when accessed from EL3. It is UNDEFINED from all other EL.
- MPAM2_EL2.TRAPMPAM1EL1: All references to it behave as if it == 0.
- MPAM2_EL2.TRAPMPAM0EL1: All references to it behave as if it == 0.
- MPAM1_EL12 is UNDEFINED when accessed from any EL.

- MPAMHCR_EL2 is RES0 when accessed from EL3.
- MPAMHCR_EL2.TRAP_MPAM_IDR_EL1: All references to it behave as if it == 0.
- MPAMHCR_EL2.GSTAPP_PLK: All references to it behave as if it == 0.
- MPAMHCR_EL2.EL1_VPMEN: All references to it behave as if it == 0.
- MPAMHCR_EL2.EL0_VPMEN: All references to it behave as if it == 0.

- MPAMVPMV_EL2 is RES0 when accessed from EL3.
- MPAMVPM0_EL2 through MPAMVPM7_EL2 are RES0 when accessed from EL3.

7.8.3 Effects if EL2 is implemented only in Non-secure state, or if implemented but disabled by SCR_EL2.EEL2 = 0 in Secure state

- MPAM2_EL2 is RW when accessed from EL3 or from Non-secure EL2. This register is UNDEFINED from all other EL.
- MPAM2_EL2.TRAPMPAM1EL1: All references to it behave as if it == 0 in the Secure state.
- MPAM2_EL2.TRAPMPAM0EL1: All references to it behave as if it == 0 in the Secure state.
- MPAM1_EL12 is RW from EL3 or from NS_EL2 when HCR_EL2.E2H == 1. This register is UNDEFINED when accessed from EL1 or EL0 or when HCR_EL2.E2H == 0.

- MPAMHCR_EL2 is RW when accessed from EL3 or from Non-secure EL2. This register is UNDEFINED from all other EL.
- MPAMHCR_EL2.TRAP_MPAM_IDR_EL1: All references to it behave as if it == 0 in the Secure state.
- MPAMHCR_EL2.GSTAPP_PLK: All references to it behave as if it == 0 in the Secure state.
- MPAMHCR_EL2.EL1_VPMEN: All references to it behave as if it == 0 in the Secure state.
- MPAMHCR_EL2.EL0_VPMEN: All references to it behave as if it == 0 in the Secure state.

- MPAMVPMV_EL2 is RW when accessed from EL3 or from Non-secure EL2. This register is UNDEFINED from all other EL.
- MPAMVPM0_EL2 through MPAMVPM7_EL2 are RW when accessed from EL3 or Non-secure EL2. These registers are UNDEFINED from all other EL.

If an implementation supports Secure state and Secure EL2 does not exist, all behaviors listed on page 7-114 must be followed by the MPAM implementation on the Secure side.If SCR_EL3.EEL2 == 0, secure EL2 behaves as if it is not implemented, and all behaviors listed on page 7-114 must be followed by the MPAM implementation on the Secure side.If Non-secure EL2 exists, the behaviors on page 7-114 do not apply to the MPAM implementation on the Non-secure side.

Chapter 8
MPAM in MSCs

This chapter contains the following sections:

8.1 Introduction

This introduction to Memory-System Components (MSCs) is *informative*. Other sections are normative unless marked as *informative*.

MSCs consist of all units that handle load or store requests issued by any MPAM master. These include cache memories, interconnects, memory management units, memory channel controllers, queues, buffers, rate adaptors, etc.

An MSC could be a part of another system component. For example, a PE might contain caches, which are MSCs. An MSC has resources that are used to process memory requests. The use of a resource could be controlled. A resource that can be controlled according to the PARTID of memory requests is partitioned. A resource might be monitored by a resource usage monitor.

8.2 Resource controls

This section is *normative*.

An MSC optionally contains one or more MPAM resource controls. Although resource controls that control different performance resources have different control parameters, all resource controls are similar in the following aspects that form a common framework:

- Each resource control uses the MPAM PARTID and MPAM_NS signals from the incoming request to select control parameters from an array of Non-secure parameters (when MPAM_NS == 1) or Secure parameters (when MPAM_NS == 0).
- The selected parameters control the behavior of the MSC, either to partition the performance resources or to control the monitoring of performance resource usage.

See *Model of a resource partitioning control* on page 5-48 for a model of a resource partitioning control. See Chapter 9 *Resource Partitioning Controls* for more detailed information on resource partitioning controls.

8.3 Security in MSCs

MPAM behavior in an MSC is affected in the following ways:

- Certain memory-mapped registers are only accessible from Secure address space (NS == 0).
- PARTIDs communicated to the MSC are augmented with a single MPAM_NS bit as 0, indicating that the MPAM PARTID in the request is to be interpreted in the Secure PARTID space. This is true even if the access from Secure state software was to the Non-secure (NS == 1) address space. MPAM_NS is always 0 if the PE is in the Secure state when the request is made, but the address of the request could be either a Secure or a Non-secure address. If the PE is in the Non-secure state, both the MPAM_NS bit and the address NS bit must be 1. See *PARTID spaces and properties* on page 3-30.
- When an MSC receives a transaction with MPAM_NS == 0, it accesses control settings for the Secure PARTID. If it receives a request with MPAM_NS == 1 it accesses the control settings for the Non-secure PARTID space.
- When programming the control settings for a Secure partition in an MSC, the settings must be stored by an access to the configuration registers in the Secure address space (NS == 0). See *Programming configuration of MPAM settings for Secure IDs*.
- When programming the control settings for a Non-secure partition in an MSC, the settings must be stored by an access to the configuration registers in the Non-secure address space (NS == 1).

8.3.1 Programming configuration of MPAM settings for Secure IDs

Configuration parameters for a Secure PARTID or Secure MPAM monitor can only be programmed from a Secure memory access (NS == 0):

- There are Secure and Non-secure versions of the MPAMCFG_PART_SEL and MSMON_CFG_MON_SEL. These two versions are accessed at the same address, differentiated by the value of the NS bit.
- Accessing an MPAMCFG_* register with a Secure (NS == 0) request accesses the configuration of a resource control of the Secure PARTID space that is selected by the PARTID in MPAMCFG_PART_SEL_S.
- Accessing an MPAMCFG_* register with a Non-secure (NS == 1) request accesses the configuration of a resource control of the Non-secure PARTID space that is selected by the PARTID in MPAMCFG_PART_SEL_NS.

8.3.2 Using Secure and Non-secure MPAM PARTIDs

When a request is processed by an MSC with MPAM resource controls, PARTID, PMG, and MPAM_NS control the partitioning control settings used and monitoring events triggered.

The PARTID and MPAM_NS of a request select the partitioning configuration from a table of PARTID configurations for each implemented resource control. The MPAM_NS bit in the request selects between the Non-secure configuration table and the Secure configuration table. The two tables do not need to have the same size. For example, the Secure configuration table might be much smaller. Tables are not required to be power-of-two sized.

A monitoring event is triggered if the PARTID, PMG, and MPAM_NS in a request match those configured in a performance monitor.

8.4 Virtualization support in system MSCs

MSCs do not see virtual PARTIDs. The PARTID generation in a requester resolves any virtual PARTID into a physical PARTID that is communicated with the memory-system request. Therefore, MSCs only handle physical PARTIDs.

8.4.1 Hypervisor emulates guest accesses to partitioning and monitoring configurations

Accesses from a guest to the configuration registers of all MSCs, and to the System registers that configure the PE MSCs, may be emulated by the host hypervisor. This allows virtual PARTID mapping to be emulated and hypervisor policies governing resource partitioning to be applied.

Configuration and reconfiguration of control settings in MSCs are expected to be rare occurrences.

Arm recommends that an MSC's memory-mapped configuration registers (See *MPAM feature page* on page 11-155) be placed at a 64-KB-aligned address to permit an access trap on that page in the stage-2 page tables. The stage-2 access traps are taken to EL2 where the hypervisor can emulate the access.

8.5 PE with integrated MSCs

A PE might have integrated MSC behaviors. These are discovered and configured as are other MSCs. See Chapter 11 *Memory-Mapped Registers*.

8.6 System-wide PARTID and PMG widths

This section is informative.

The behavior of MSCs is UNPREDICTABLE if it receives an MPAM PARTID or PMG outside the range it supports.

For predictable behavior, the PARTID on a request by a master should be in the range of 0 to:

- If the request is MPAM_NS == 1 (to Non-secure ID spaces), the smallest maximum Non-secure PARTID supported by any MSC that might be accessed by that request.
- If the request is MPAM_NS == 0 (to Secure ID spaces), the smallest maximum Secure PARTID supported by any MSC that might be accessed by that request.

And, the PMG on a request by a master should be in the range of 0 to:

- If the request is MPAM_NS == 1 (to Non-secure ID spaces), the smallest maximum Non-secure PMG supported by any MSC that might be accessed by that request.
- If the request is MPAM_NS == 0 (to Secure ID spaces), the smallest maximum Secure PMG supported by any MSC that might be accessed by that request.

The smallest maximum values for PARTID and PMG in Non-secure and Secure spaces can be computed from firmware during discovery. PARTID and PMG widths are reported through ID registers in PEs and MSCs. See sections Appendix B *MSC Firmware Data*, *System register descriptions* on page 7-75, and *Determining presence and location of MMRs* on page 11-154.

8.7 MPAM interrupts

This section is *normative*.

There are two types of interrupts that an MPAM MSC could produce:

- MPAM Error Interrupt.
- MPAM Overflow Interrupt.

8.7.1 MPAM Error Interrupt

MPAM errors in MSCs are described in *Error conditions in accessing memory-mapped registers* on page 12-243.

MPAM errors that are detected in an MSC are recorded in MPAMF_ESR and signalled to software via an MPAM error interrupt if enabled by MPAMF_ECR.INTEN == 1.

If an MSC cannot encounter any of the error conditions listed in *Error conditions in accessing memory-mapped registers* on page 12-243, both the MPAMF_ESR and MPAMF_ECR must be RAZ/WI.

If an MSC supports both Secure and Non-secure address spaces, MPAMF_ESR and MPAMF_ECR will each have a Secure instance and a Non-secure instance. The Secure registers control and generate Secure MPAM error interrupts, while the Non-secure registers control and generate Non-secure MPAM error interrupts.

The MPAM error interrupt can be implemented in an MSC as a level-sensitive interrupt or as an edge-triggered interrupt. The interrupt behavior depends on whether level-sensitive or edge-triggered interrupts are used.

- Arm recommends that the MPAM error interrupt be implemented as a level-sensitive interrupt.
- The mechanism by which an interrupt request from an MSC resource monitor generates an FIQ or IRQ exception is IMPLEMENTATION DEFINED.
- Arm recommends that an MSC implements two MPAM error interrupt signals, one for the Secure MPAM error interrupt and another for the Non-secure MPAM error interrupt.
- Arm recommends that MPAM error interrupt requests:
  - Translate into an MPAM_ERR_IRQ signal, so that they are observable to external devices.
  - If the MSC is integrated into a PE, connect to inputs on an IMPLEMENTATION DEFINED generic interrupt controller as a Private Peripheral Interrupt (PPI) or a Locality-specific Peripheral Interrupt (LPI) for that PE. See the Arm Generic Interrupt Controller Architecture Specification for information about PPIs, LPIs, and SPIs.
  - If the MSC is not integrated into a PE, connect to inputs on an IMPLEMENTATION DEFINED generic interrupt controller as a System Peripheral Interrupt (SPI) or Locality-Specific Peripheral Interrupt (LPI).

Level-sensitive interrupts

When using level-sensitive interrupts, the interrupt is active when MPAMF_ESR.ERRCODE is non-zero.

Software can make a level-sensitive interrupt active by writing non-zero to MPAMF_ESR.ERRCODE.

An interrupt service routine is expected to write `0b0000` into MPAMF_ESR.ERRCODE to clear the interrupt.

See also Chapter 12 *Errors in MSCs*.

Edge-triggered interrupts

When using edge-triggered interrupts, the interrupt edge is generated when MPAMF_ESR.ERRCODE is written due to an error.

An edge-triggered interrupt is not generated when software writes to MPAMF_ESR.

An interrupt service routine does not need to clear an edge-triggered interrupt.

See Chapter 12 *Errors in MSCs* for other reasons for an interrupt service routine to clear MPAMF_ESR.

8.7.2 MPAM overflow interrupt

A monitor could overflow, especially if it is a type of monitor that accumulates counts. If it is possible for a type of monitor to overflow, there are bits in MSMON_CFG_*_CTL to control the behavior on overflow (*Overflow bit* on page 10-150). OFLOW_INTR == 1 causes an overflow of the counter to produce an MPAM Overflow Interrupt.

If an MSC supports both Secure and Non-secure address spaces, MSMON_CFG_*_CTL registers and MSMON_MBWU and MSMON_CSU registers that are implemented have Secure and Non-secure instances. Secure instances of MSMON_CFG_*_CTL.OFLOW_INTR control whether a Secure MPAM overflow interrupt is generated when the corresponding Secure counter instance overflows. Non-secure instances of MSMON_CFG_*_CTL.OFLOW_INTR control whether a Non-secure MPAM overflow interrupt is generated when the corresponding Non-secure counter instance overflows.

- The mechanism by which an interrupt request from an MSC resource monitor generates an FIQ or IRQ exception is IMPLEMENTATION DEFINED.
- Arm recommends that an MSC implements two MPAM overflow interrupt signals, one for the Secure MPAM overflow interrupt and another for the Non-secure MPAM overflow interrupt.
- Arm recommends that MPAM overflow interrupt requests:
  - Translate into an MPAM_OF_IRQ signal, so that they are observable to external devices.
  - If the MSC is integrated into a PE, connect to inputs on an IMPLEMENTATION DEFINED generic interrupt controller as a Private Peripheral Interrupt (PPI) or a Locality-specific Peripheral Interrupt (LPI) for that PE. See the Arm Generic Interrupt Controller Architecture Specification for information about PPIs, LPIs and SPIs.
  - If the MSC is not integrated into a PE, connect to inputs on an IMPLEMENTATION DEFINED generic interrupt controller as a System Peripheral Interrupt (SPI) or Local Peripheral Interrupt (LPI).

The overflow interrupt is a level-sensitive interrupt. The interrupt is reset by writing 0 to the OFLOW_STATUS field of all overflowed monitor instances MSMON_CFG_*_CTL register.

Chapter 9
Resource Partitioning Controls

This chapter contains the following sections:

9.1 Introduction

This introduction to memory-system partitioning is *informative*. Other sections are *normative* unless marked as *informative*.

Software assigns VMs and applications to a partition. The hypervisor can assign VMs to partitions, and operating systems can assign applications to partitions. This specification does not address how such assignments are made by software.

A memory-system partition is associated with a software environment on a PE by loading an MPAMn_ELx register with PARTID_I and PARTID_D. An EL2 hypervisor loads MPAM1_EL1 with the partition IDs when context-switching between VMs. An EL1 operating system loads MPAM0_EL1 with the partition IDs when context-switching between applications. The PARTIDs loaded into fields of MPAMn_ELx for instruction and data accesses are used for requests when running software at ELn. The PARTID on memory-system requests connects the software environment to the resource partitioning controls in the MSCs that handle the requests.

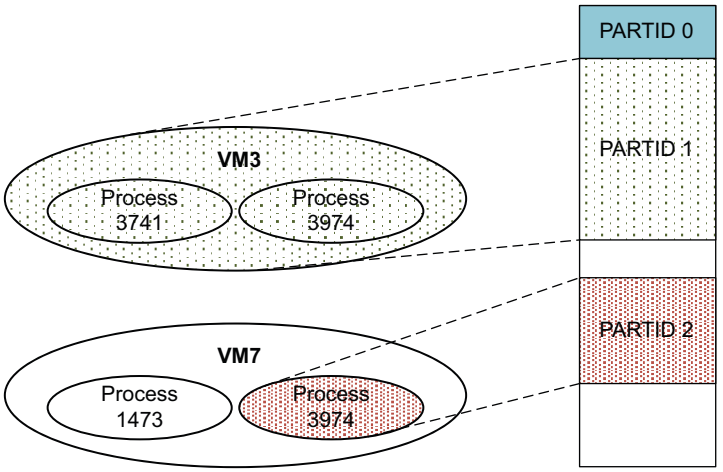


Figure 9-1 Partitioning, VMs, and OS processes

The PARTID of a request controls uses of each MSC's performance resources. An MSC receives a PARTID with each request. The PARTID may be used within the component to select resource controls for the component's resource allocation and utilization behavior.

All memory-system requests with a given PARTID share the resource control settings for that partition.

Because a PARTID is communicated to shared MSCs and interpreted there, PARTIDs should be managed and allocated on a system-wide basis.

Resource partitioning controls might be standard or implementation specific.

Standard control interfaces are architected, but optional. Therefore, an MSC that does not require a standard control interface does not need to implement it. Most MSCs implement few of the standard control interfaces.

An implementation-specific resource control can use a PARTID for unique facilities that either control resources not envisioned by the standard controls or that implement unique control methods that cannot be mapped onto the standard control interfaces.

9.2 Partition resources

An MSC contains resources that affect the performance of the memory system. For such a resource to be partitionable:

- The component must support MPAM at its upstream interface.
- The component must have one or more MPAM resource controls for that resource.

A partitionable resource may be partially allocated to a partition according to the programming of the MPAM resource control or controls for that resource.

9.3 Standard partitioning control interfaces

The MPAM architecture defines standard partitioning control interfaces. This enables binary distribution of operating systems supporting MPAM.

The MPAM architecture defines the following standard types of control interfaces for memory-system resources:

- Cache-portion partitioning.
- Cache maximum-capacity partitioning.
- Memory-bandwidth portion partitioning.
- Memory-bandwidth minimum and maximum partitioning.
- Memory-bandwidth proportional-stride partitioning.
- Priority partitioning.

Each of these standard control interfaces is optional at each MSC. An MSC may implement several controls or none. Some controls only make sense for certain types of MSCs, or for certain implementations of an MSC. Others may be possible but too costly for the system's target market.

Cache-portion partitioning and memory-bandwidth portion partitioning follow the generic portion-control interface described in *Portion resource controls* on page A-259. Cache maximum-capacity partitioning follows the generic maximum-usage control interface described in *Maximum-usage resource controls* on page A-260.

The presence of each standard control is indicated by a bit in the MPAMF_IDR memory-mapped ID register for that component, or in a resource-specific memory mapped ID register. See *Memory-mapped ID register description* on page 11-161.

The standard partitioning control interfaces share a PARTID selection register, MPAMCFG_PART_SEL (Chapter 11). To program a control, first set MPAMCFG_PART_SEL to the PARTID for which you require to set a control value. Then, set the control's configuration register (MPAMCFG_*) to the new control value.

Software must ensure mutual exclusion for access to MPAMCFG_* registers of each MSC.

9.3.1 Cache-portion partitioning

A portion is a uniquely identifiable part of a resource. It is of fixed size or capacity and all portions of a resource are the same size. A particular resource has a constant number of portions. Every partition that is given access to a portion n shares access to portion n.

The storage portions of caches may be partitioned. Allocating portions of a cache to a partition permits requests attributed to that partition to allocate within those portions of the cache.

When a request to a cache requires a cache line to be installed in the cache, the PARTID of that request determines which portions of the cache the request may allocate to install the line.

Cache-portion partitioning uses the generic portion-partitioning interface described in *Portion resource controls* on page A-259.

Cache-portion bit map

A cache-portion bitmap (CPBM) controls the cache-storage portion allocation for a partition. Each bit of a CPBM controls whether the partition is permitted to allocate a particular capacity portion of the cache. The number of capacity portions available in a cache is an IMPLEMENTATION DEFINED parameter that is discoverable in MPAMF_CPOR_IDR for the cache. The width of the CPBM field is equal to the number of capacity portions available in the cache.

For example, assume a cache has a 1 MB total capacity in 32 portions. Each portion has a capacity of 1 MB / 32 = 32 KB. A partition has 4 portions allocated (only 4 bits in the CPBM are 1's). So, this partition can only allocate into these particular 4 portions, allowing up to 128 KB, or 1/8th of the cache's total capacity.

CPBM is an instance of the generic portion bitmap (PBM) described in *Portion resource controls* on page A-259.

Over-allocation of capacity portions

Storage capacity portions cannot be over-allocated. This is true because the CPBM contains bits that control allocations in the implementation-dependent number of allocable capacity portions of the cache.

Changing CPBM for a partition

Software may change the CPBM during system operation. This does not disrupt normal system operation because the CPBM only affects new allocations and does not reallocate previously allocated cache storage.

If a cache line was allocated under a previous CPBM to a portion that is not set in the new CPBM, the partition is using more of the cache capacity than it is entitled to under the new CPBM:

- If lines previously allocated in a portion that is not in the new CPBM are not accessed again, they will eventually be reallocated to a partition that has its CPBM bit set for that portion of the capacity. So, these will represent a temporary misallocation of capacity.
- If however, a line that is present in the cache in a portion that is not in the new CPBM continues to be accessed, this can lead to a long-term misallocation of capacity. The line's location optionally might be updated, see *Write hits that update the PARTID of a cache line may move that line to a different portion* on page 5-51.

Using cache-portion partitioning with cache maximum-capacity partitioning

When cache-portion partitioning is used with cache maximum-capacity partitioning, both controls are effective as described in *Using cache maximum-capacity partitioning with cache-portion partitioning* on page 9-132.

9.3.2 Cache maximum-capacity partitioning

A limit may be set on the storage capacity of a cache that a memory-system partition may use. Setting a maximum cache capacity to a partition permits requests attributed to that partition to allocate up to that maximum cache capacity. Attempts to allocate beyond that capacity must limit a partition's capacity usage.

(informative) Examples of techniques for limiting cache usage by a new request when a partition's capacity usage is at or above its maximum include:

- Do not allocate for the new request.
- Replace some data from that partition with data from the new request.
- Evict some data from that partition from the cache before allocating for the new request.
- Defer the required deallocation until a more convenient time.

Cache lookups are not affected by partitioning. A cache lookup must find a valid cache line even if that line was allocated with a different PARTID.

Cache maximum-capacity partitioning follows the description of the generic maximum-usage resource control interface described in *Maximum-usage resource controls* on page A-260.

Cache maximum-capacity control setting

The cache maximum-capacity control setting is programmed by storing a capacity limit into the MSC's cache maximum capacity control interface, MPAMCFG_CMAX.

The cache maximum-capacity limit is a fraction of the cache's total capacity. The format of the limit value is a fixed-point fraction, as described in *About the fixed-point fractional format* on page 9-142.

For example, to allocate 30% of a 256 KB cache to a partition:

- In the fixed-point fractional format, 1.0 is represented as 216 – 1, or in hex as 0xFFFF. The subtraction makes 1.0 within the range of the representation.

- So, the representation of 30% would be 1.0 * 0.30, which in hex is 0xFFFF * (decimal) 0.30, or 0x4CCC.
  - Similarly, 25% would be 0x3FFF; 14% would be 0x23D6; 3% would be 0x07AE; and 3.25% would be 0x0851.
- If you have a cache with 256 KB of capacity, and the resource control setting for a PARTID is set to 0x4CCC to represent 30%, that partition is permitted to use 30% of the cache, or about 76.75 KB of capacity.
- Since most, but not all, Arm caches have 64-byte lines, a 256 KB cache has 4096 of these 64-byte lines, and 30% of those lines is 1228 or 76.75 KB.

The fixed-point fractional format permits an implementation to leave bits to the right as unimplemented, meaning that the value would be truncated to the implemented bits, causing some of the right-most bits to be zeros:

- As an example, the 3% value previously mentioned is 0x07AE. If only 8 bits of fraction are implemented, when software stores 0x07AE into a resource control setting, the value is shortened to the most significant bits and stored as 0x07--.
- When using the resource control setting, the unimplemented bits would be read as zeros.

The actual value of the setting is therefore an interval from the value of the control setting up to the value of the control setting plus one in the right-most implemented bit.

- In the case of the 3% value previously mentioned, that interval is from 0x07 (2.734%) to 0x08 (3.125%).
- An implementation is permitted to regulate the resource to any point within this interval.

Using cache maximum-capacity partitioning with cache-portion partitioning

When cache-portion partitioning is used with cache maximum-capacity partitioning, both controls are effective. Cache-portion partitioning controls which portions of the capacity may be allocated to this partition. Cache maximum-capacity partitioning limits the amount to less than or equal to a cache-capacity limit control setting.

For example, assume several portions of the capacity are shared by several partitions. Any such partition can allocate within the shared portions. To keep one of the partitions from using too much of the shared allocation, the maximum-capacity controls for the partitions can each be set to less than the capacity of the portions to which they may allocate. If each partition is given 50% of the capacity of the shared portions, then no one partition can use more than 50% of the shared cache portions.

Here is an example of a cache with 1 MB total capacity in 32 portions. Each partition has 4 portions for shared allocation. To allow a partition to use no more than 50% of its shared allocation, you would set the cache maximum-capacity limit for this partition as follows:

1. Portions divide the capacity of the cache into distinct parts of the same size. So, for a 1 MB cache divided into 32 portions, each portion has 1 MB / 32 = 32 KB:
   a. In portion partitioning, it is not possible to allocate anything other than an integral number of portions to a PARTID.
   b. A cache portion may be exclusively allocated to a PARTID or it may be shared by 2 or more PARTIDs.
   c. A PARTID that has 4 portions allocated to it is permitted to use 32 KB * 4 = 128 KB.
2. The combined behavior of cache-portion partitioning and cache maximum-capacity control has both controls:
   a. To allow a PARTID to use only 50% of the storage in the portions allocated to it, the cache maximum-capacity control is used.
   b. Compute the fraction of the cache that is 50% of the storage in the portions allocated. In this case, it is 64 KB / 1 MB = 1/16 or 6.25%, which is 0x0FFF in the fixed-point fractional representation.
   c. The combined behavior only permits the PARTID to allocate storage in the 4 portions it may use according to the cache-portion control, but its use of storage is also limited to 50% of the storage of those portions.

Over-allocation of capacity

Cache capacity can be over-allocated because the sum of the cache-capacity control parameters may exceed 100% of the cache size. This may be acceptable. The cache-capacity control does not provide a minimum cache capacity guarantee, only a maximum guarantee. The data of inactive partitions may be evicted from the cache due to the activity of other partitions.

9.3.3 Memory-bandwidth portion partitioning

An MSC's downstream bandwidth may be divided into portions, and those portions may be allocated to partitions.

Memory-bandwidth portion partitioning follows the generic portion-control interface described in *Portion resource controls* on page A-259, in which a portion is a quantum of bandwidth. A Time-Division Multiplexing (TDM) scheme that allocates traffic to time slots is an example of a bandwidth allocation system that has portions.

The BandWidth Portion Bit Map (BWPBM) is the Portion Bit Map (PBM) for bandwidth.

9.3.4 Memory-bandwidth minimum and maximum partitioning

An MSC's downstream bandwidth may be partitioned by bandwidth usage. There are two bandwidth-usage control schemes. An MSC can optionally implement each of them:

- Minimum bandwidth to which the PARTID has claim, even in the presence of contention.
- Maximum bandwidth limit available to the PARTID, in the presence of contention.

The minimum and maximum bandwidth partitioning schemes rely on tracking bandwidth usage by PARTIDs. Because bandwidth is measured in bytes per second, bandwidth measurements have a dependence on time. That dependence is captured in this specification as the accounting window or accounting period. See *Memory-bandwidth allocation accounting window width* on page 9-135

Without contention, the bandwidth may be strictly limited to the maximum or permitted to use more than the maximum, since no other partition's traffic is claiming that bandwidth.

Any combination of these control schemes may be used simultaneously in an MSC that supports them.

Each control scheme is described below.

Minimum-bandwidth limit partitioning

The minimum-bandwidth control scheme regulates the bandwidth used by a PARTID's requests:

- If the bandwidth usage by the PARTID of the request, as tracked during the accounting period, is currently less than the partition's minimum, its requests are preferentially selected to use downstream bandwidth.
- If the bandwidth usage by the PARTID of the request, as tracked during the accounting period, is currently greater than or equal to the PARTID's minimum, its requests compete with other requests as described under *Maximum-bandwidth limit partitioning* on page 9-134, if implemented. If maximum-bandwidth limit partitioning is not implemented, requests with PARTID that have current bandwidth usage greater than that PARTID's minimum-bandwidth limit compete with all requests and do not receive preferential treatment under the minimum-bandwidth limit.

A PARTID's requests below its minimum bandwidth are therefore most likely to be scheduled to use downstream bandwidth.

Bandwidth that is not used by a partition during an accounting window does not accumulate.

The control parameter is a fixed-point fraction of the available bandwidth. See *About the fixed-point fractional format* on page 9-142.

Maximum-bandwidth limit partitioning

The maximum-bandwidth limit control scheme regulates the bandwidth used by a PARTID's requests:

- If the bandwidth usage by the PARTID as tracked during the accounting period is currently less than the PARTID's maximum bandwidth but greater than or equal to its minimum bandwidth, if implemented, its requests are selected to use bandwidth when there are no competing minimum bandwidth requests to service. Requests for PARTIDs that are above their minimum-bandwidth limits but less than their maximum-bandwidth limits compete with each other to use bandwidth.
- If the bandwidth usage by the PARTID of the request is greater than or equal to the PARTID's maximum bandwidth and the HARDLIM bit is not set, the request competes with other such requests to use bandwidth when there are no competing requests to service for PARTIDs currently below their minimum bandwidth or maximum bandwidth.
- If the bandwidth usage by the PARTID of the request is greater than or equal to the PARTID's maximum bandwidth and the Hard Limit (HARDLIM) bit is set, the requests are saved until the PARTID's bandwidth usage drops below its maximum bandwidth control setting.

If the HARDLIM bit is set, the partition is prevented from using more bandwidth if the current bandwidth usage is over the maximum bandwidth limit. As the accounting window advances, the current bandwidth usage resets to zero or otherwise decays, permitting the partition to again use bandwidth.

Bandwidth that is not used by a partition during an accounting window does not accumulate.

The control parameter is a fixed-point fraction of the available bandwidth. See *About the fixed-point fractional format* on page 9-142.

Using minimum-bandwidth limit with maximum-bandwidth limit controls

If both minimum-bandwidth limit and maximum-bandwidth limit are implemented, Table 9-1 shows the preference of requests.

Table 9-1 Preference of requests for bandwidth limits

If used bandwidth is		The preference is	Description
Below the minimum		High	Only other High requests delay this request*.
Above the minimum	Below the maximum limit.	Medium	High requests are serviced first, then compete with other Medium requests*.
	Above the maximum limit, with HARDLIM clear.	Low	Requests are not serviced if any High or Medium requests are available*.
	Above the maximum limit, with HARDLIM set.	None	Requests are not serviced.

* Implementations may occasionally deviate from preference order in servicing requests to meet other goals, such as starvation avoidance.

Bandwidth control parameters

The control parameters for bandwidth partitioning schemes are all expressed in a fixed-point fraction of the available bandwidth. See *About the fixed-point fractional format* on page 9-142.

The bandwidth control setting register for maximum-bandwidth limit (MPAMCFG_MBW_MAX) also includes a Hard Limit (HARDLIM) bit that prevents a partition from using more than the maximum fraction of the available bandwidth that is set in that register.

Memory-bandwidth allocation accounting window width

For both the minimum- and maximum-bandwidth partitioning schemes, memory-bandwidth regulation occurs over an accounting window. The accounting may be either a moving window or by resetting bandwidth counts at the beginning of each accounting-window period.

The width of the window is discoverable and can be read from MPAMCFG_MBW_WINWD for the PARTID selected by MPAMCFG_PART_SEL.

In implementations that support settable window width per PARTID, MPAMCFG_MBW_WINWD may be written with a fixed-point format (as described in the register's description) specifying the accounting window width in microseconds.

Fixed accounting window

In fixed-window accounting, bandwidth is apportioned to requests so that each partition gets bandwidth according to the minimum and maximum for that partition (*Over-allocation of minimum bandwidth*). Request or local priorities (*Priority partitioning* on page 9-136) are used to resolve conflicting requests of the same preference.

When the accounting window's period is reached, a new window begins with no history except for any queue of requests that have not been previously serviced. The new window starts accumulating bandwidth for a partition from zero.

Moving-window accounting

A moving window tracks partition bandwidth usage by all commands issued in the past window width. There is never a reset of the accounting of bandwidth usage per partition. Instead, bandwidth is added to the accounting when a command is processed and removed from the accounting when that command moves out of the window's history. This continuous accounting is relatively free from boundary effects.

Moving-window accounting requires hardware to track the history of commands within the window, in addition to the bandwidth counters per PARTID required by the fixed window.

Other accounting window schemes

An implementation may use another scheme for maintaining history that is broadly in line with the schemes described here. For example, the current bandwidth might decay at a fixed rate proportional to the bandwidth allocation, but not below a current bandwidth of zero.

Over-allocation of minimum bandwidth

The minimum bandwidth allocations of all partitions may sum to more bandwidth than is available. This is not a problem when some partitions are not using their bandwidth allocations, because unused allocations are available for other partitions to use. However, when minimum bandwidth is over-allocated, the minimum bandwidth that is programmed for partitions cannot always be met.

If the programmed minimum bandwidth allocation is to be reliably delivered by the system, software must ensure that minimum bandwidth is not over-allocated.

Over-allocation of maximum bandwidth

The maximum bandwidth allocations of all partitions may sum to more bandwidth than is available. This is not a problem when some partitions are not using their maximum bandwidth allocations, because unused allocations are available for other partitions to use. If maximum bandwidth is over-allocated, the maximum bandwidth that is programmed for partitions cannot always be met.

Available bandwidth

The bandwidth available downstream from an MSC is not constant, and it affects the operation of minimum and maximum bandwidth partitioning.

Available bandwidth may depend on one or more clock frequencies in many systems (for example, DDR clock). Software may require to reallocate bandwidths when changing clock frequencies that affect available bandwidth. Lowering clock rates without changing allocations may result in over-allocation of bandwidth.

The available bandwidth on a DRAM channel varies with the mix of reads and writes and the bank-hit rate. Bandwidth may also vary with burst size.

9.3.5 Memory-bandwidth proportional-stride partitioning

Proportional-stride bandwidth partitioning control is an instance of proportional resource-allocation generic control, described in *Proportional resource allocation facilities* on page A-261. The control parameter for bandwidth proportional-stride partitioning is expressed as an unsigned integer.

Regulation according to this scheme permits the partition to consume bandwidth in proportion to its stride, in relation to other requests' strides that are contending for bandwidth. See *Model of stride-based memory bandwidth scheduling* on page A-261 for an example of stride-based proportional bandwidth regulation.

The MPAMF_MBW_IDR.HAS_PROP bit indicates the presence of a memory-bandwidth proportional-stride partitioning control interface in the MSC.

Combining memory-bandwidth proportional stride with other memory-bandwidth partitioning

There is no setting of the STRIDEM1 control field that disables the effects of proportional-stride partitioning on a partition's bandwidth usage. To enable proportional-stride partitioning for a PARTID, MPAMCFG_MBW_PROP.EN must be set to 1.

When multiple partitioning controls are active, each affects the partition's bandwidth usage. However, some combinations of controls may not make sense, because the regulation of that pair of controls cannot be made to work in concert.

Memory-bandwidth maximum partitioning must work together with proportional-stride partitioning.

9.3.6 Priority partitioning

Unlike the other memory-system resources in this architecture, priority does not directly affect the allocation of memory-system resources. Instead, it has an effect on conflicts that arise during access to resources. A properly configured system should rarely have substantial performance effects due to prioritization, but priority does play an important role in oversubscribed situations, whether instantaneous or sustained. Therefore, we choose to include priority partitioning here as a tool to aid in isolating memory-system effects between partitions.

A PARTID may be assigned priorities for each component in the memory system that implements a priority partitioning control. This partitioning control allows different parts of the memory system to handle requests with different priorities. For example, requests from a PE to system cache may be set to have a higher transport priority than those from system cache to main memory.

In a system in which the interconnect carries QoS values or priorities, requests arriving at an MSC have an upstream priority as part of the request. In the absence of an internal priority partitioning control, request priority could be used by an MSC to prioritize internal operations. In the absence of a downstream priority partitioning control, the request priority is used as through priority. See *Through priorities* on page 9-137.

Priority partitioning can override the upstream priority with two types of priorities:

- Internal priorities control priorities used in the internal operation of an MSC.
- Downstream priorities control priorities communicated downstream (for example to an interconnect).

"Downstream" refers to the communication direction for requests. "Upstream" refers to the response, and it usually uses the same transport priority as the request that generated it.

Internal priorities

Internal priorities are used within an MSC to prioritize internal operations. For example, a memory controller may use an internal priority to choose between waiting requests when bandwidth allocation indicates two or more requests have the same bandwidth preference.

Internal priority partitioning is optional even if downstream priority partitioning is implemented.

Downstream priorities

An MSC uses a downstream priority to set transport priorities for downstream requests generated during the servicing of an incoming request from upstream.

Downstream priority partitioning is optional even if internal priority partitioning is implemented.

Through priorities

For a system in which the interconnect carries QoS values or priorities, these priorities arrive with incoming requests from upstream. An MSC that does not implement priority partitioning, or that does not implement downstream priority partitioning, must use these upstream priorities on all downstream communication.

If an MSC does not implement priority partitioning, or it does not implement downstream priorities, the downstream priority is always the same as the request (upstream) priority.

The priority of a response through an MSC (from downstream to upstream) is always the same priority as the response received (from downstream). Priority partitioning never alters response priorities received from downstream.

9.4 Vendor or implementation-specific partitioning control interfaces

MPAM provides discoverable vendor extensions to permit partners to invent partitioning controls. These include controls that do not fit the standard interfaces and controls for types of resources not supported through the standard controls defined in this document. Such controls provide product differentiation to address market-segment needs or to provide superior memory-system control.

The MPAMF_IDR.HAS_IMPL_IDR bit indicates the presence of MPAMF_IMPL_IDR and of implementation-specific or vendor-specific resource partitioning controls.

Vendor, design, or model and version information is present in MPAMF_IIDR. MPAMF_IMPL_IDR is available for implementations that need to convey additional information about parameters of implementation-specific partitioning controls.

9.5 Measurements for controlling resource usage

This section is *informative*.

In many cases, resource usage by a partition must be measured so that the resource controller can regulate allocation of the resource to that partition.

In a memory channel, the bytes delivered to requests from a PARTID might be more costly if delivered in response to a series of 1-byte requests rather than cache-line-sized bursts. So, it might be reasonable to count the cost of servicing a 1-byte request to be the same as the cost of servicing a cache-line request rather than as a fraction of a word access cost.

9.6 PARTID narrowing

An implementation may optionally map input PARTID spaces into smaller internal PARTID spaces. This involves mapping the PARTID from a request (reqPARTID) into an internal PARTID (intPARTID). The reqPARTID-to-intPARTID mappings for Secure and Non-secure physical PARTID spaces must be used internally and not for downstream requests.

This mapping is supported by a memory-mapped register, MPAMCFG_INTPARTID, and an ID register bit for each of the Secure and Non-secure physical PARTID spaces. The related behavior includes:

- Translate the incoming request's reqPARTID and MPAM_NS into an intPARTID (with the same MPAM_NS) before accessing the control settings and regulation state of the partition.
- Use MPAMCFG_INTPARTID to store an association of a reqPARTID in MPAMCFG_PART_SEL to the intPARTID stored in MPAMCFG_INTPARTID.
- Error code for MPAMF_ESR to indicate a bad intPARTID mapping for the reqPARTID.
- A bit in MPAMCFG_PART_SEL indicates that the value in that register is an intPARTID. The register can hold either an intPARTID or reqPARTID at any time, but the reqPARTID can only be used for accessing the association by means of MPAMCFG_INTPARTID. So, at the time MPAMCFG_INTPARTID is read or written, MPAMCFG_PART_SEL.INTERNAL must be clear. For access to read or write other control settings registers, the INTERNAL bit must be set.
- With PARTID narrowing implemented, the contents of MPAMCFG_PART_SEL are interpreted as an intPARTID for accessing control settings through an MPAMCFG_* register other than MPAMCFG_INTPARTID. The MPAMCFG_PART_SEL.INTERNAL bit must be set to confirm the intPARTID is being used.
- With PARTID narrowing not implemented, the contents of MPAMCFG_PART_SEL are interpreted as a reqPARTID. The MPAMCFG_PART_SEL.INTERNAL bit must == 0 to confirm that the reqPARTID is being used.

9.7 System reset of MPAM controls in MSCs

This section is *normative*.

After a system reset, the MPAM controls in MSCs must reset the settings for default PARTID (*Default PARTID* on page 3-30) so that software can use all of the resource. Since MPAMn_ELx.MPAMEN for the highest implemented ELx is reset to 0 by a system reset, the MPAM fields of all requests issued by a PE use the corresponding default PARTID in the PE's current Security state. Only the resource controls for the default PARTIDs must be reset to full access for the system to behave as if there were no MPAM.

Only the control settings for the default PARTID must be reset. The reset value should be appropriate to allow the default PARTID to access all of the resource. This is needed to allow the system to boot up to a point where MPAM resource controls can be set before non-default PARTIDs are used to make requests.

9.7.1 Suggested reset values for standard control types

Table 9-2 shows the suggested reset values for PARTID == 0 control setting for both MPAM_NS == 0 and MPAM_NS == 1.

Table 9-2 Suggested reset values for standard control types

Control type	Reset value
MPAMCFG_CPOR	all ones
MPAMCFG_CMAX	0xFFFF
MPAMCFG_MBW_PBM	all ones
MPAMCFG_MBW_MAX	0xFFFF
MPAMCFG_MBW_PROP	EN=0

In addition, for PARTID narrowing, Arm suggests that reqPARTID == 0 map to intPARTID == 0 and that the reset values be applied to the settings of intPARTID == 0 in both values of MPAM_NS.

9.8 About the fixed-point fractional format

This section is *normative*.

Fractional control parameters use a 16-bit fixed-point format. The format permits implementations to have fewer than 16 bits by truncating least significant bits from the fraction and implementing these bits as RAZ/WI.

Software can be expected to calculate a 16-bit fractional part to store into the memory-mapped register without the need to understand the implemented width of the field. If the field width is less than 16 bits, the least significant bits are silently IGNORED by the implementation. This results in an uncertainty of the intended value.

If software stores an intended fractional value into a field with an implemented width of w, the implementation's truncated field sees a value of v. The value v is at the bottom of the range of v to $v + 2^{-w} - 2^{-17}$ and the intended fractional value lies somewhere within that range, inclusive of the end points.

Depending on the use of the fractional value, the best choice of value within the range could be the center of the range, the smallest end of the range, or the greatest end of the range. For examples, a cache maximum-capacity fraction might best be interpreted as the highest end of the range, and a cache minimum-capacity fraction might best be interpreted as the lowest end of the range.

Table 9-3 shows the fraction widths and hex representation used for three formats. The values in the table are suitable for a maximum limit because the Max value for every entry is never greater than the target value.

Table 9-3 Fraction Widths and Hex Representation

Percentage	16 bits			12 bits			8 bits		
	Hex	Min	Max	Hex	Min	Max	Hex	Min	Max
1.00%	028E	0.9979%	0.9995%	027	0.9521%	0.9766%	01	0.3906%	0.7813%
12.50%	1FFF	12.4985%	12.5000%	1FF	12.4756%	12.5000%	1F	12.1094%	12.5000%
16.67%	2AAB	16.6672%	16.6687%	2A9	16.6260%	16.6504%	29	16.0156%	16.4063%
25%	3FFF	24.9985%	25.0000%	3FF	24.9756%	25.0000%	3F	24.6094%	25.0000%
33.33%	5552	33.3282%	33.3298%	554	33.3008%	33.3252%	54	32.8125%	33.2031%
35%	5998	34.9976%	34.9991%	598	34.9609%	34.9854%	58	34.3750%	34.7656%
37.25%	5F5B	37.2482%	37.2498%	5F4	37.2070%	37.2314%	5E	36.7188%	37.1094%
42.50%	6CCB	42.4973%	42.4988%	6CB	42.4561%	42.4805%	6B	41.7969%	42.1875%
45%	7332	44.9982%	44.9997%	732	44.9707%	44.9951%	72	44.5313%	44.9219%
50%	7FFF	49.9985%	50.0000%	7FF	49.9756%	50.0000%	7F	49.6094%	50.0000%
52%	851D	51.9974%	51.9989%	850	51.9531%	51.9775%	84	51.5625%	51.9531%
55%	8CCB	54.9973%	54.9988%	8CB	54.9561%	54.9805%	8B	54.2969%	54.6875%
58%	9479	57.9971%	57.9987%	946	57.9590%	57.9834%	93	57.4219%	57.8125%
62.75%	A0A2	62.7472%	62.7487%	A09	62.7197%	62.7441%	9F	62.1094%	62.5000%
66.67%	AAA9	66.6641%	66.6656%	AA9	66.6260%	66.6504%	A9	66.0156%	66.4063%
75%	BFFF	74.9985%	75.0000%	BFF	74.9756%	75.0000%	BF	74.6094%	75.0000%
82.50%	D332	82.4982%	82.4997%	D32	82.4707%	82.4951%	D2	82.0313%	82.4219%
88%	E146	87.9974%	87.9990%	E13	87.9639%	87.9883%	E0	87.5000%	87.8906%
95%	F332	94.9982%	94.9997%	F32	94.9707%	94.9951%	F2	94.5313%	94.9219%
100%	FFFF	99.9985%	100.0000%	FFF	99.9756%	100.0000%	FF	99.6094%	100.0000%
2^n	65536			4096			256		
ndigits	4			3			2		
shift	0			0			0		

Chapter 10
Resource Monitors

This chapter contains the following sections:

10.1 Introduction

Software environments may be labeled as belonging to a Performance Monitoring Group (PMG) within a partition. The PARTID and PMG can be used to filter some performance events so that the performance of a particular PARTID and PMG can be monitored.

10.2 MPAM resource monitors

MPAM resource monitors provide software with measurements of the resource-type usage that can be partitioned by MPAM. There are two types of MPAM resource monitors:

- Memory-bandwidth usage.
- Cache-storage usage.

Each type of monitor measures the usage by memory-system transactions of a PARTID and PMG. An MSC may implement any number of performance monitor instances, up to 216 of each type.

To access a monitor instance, the instance number is stored into the MSMON_CFG_MON_SEL.MON_SEL field. All of the monitor access registers for a type of monitor then access that instance of that type. See *Monitor configuration* on page 10-151.

10.2.1 Memory-bandwidth usage monitors

A memory-bandwidth usage monitor counts payload bytes meeting the filter criteria that pass the monitoring point in the downstream direction for writes or the upstream direction for reads. Each monitor has the following set of memory-mapped configuration registers and functional features:

- A control register (MSMON_CFG_MBWU_CTL) configures behavior of the monitor instance.
- A filter register (MSMON_CFG_MBWU_FLT) specifies the transfers to be counted. This register has fields for reads, writes, PARTID, PMG, and other criteria.
- A monitor register (MSMON_MBWU) contains an optionally scaled count of bytes transferred downstream from this MSC that match the conditions of the filter register. This monitor register may be reset after each capture event. If scaling is enabled, the value read from MSMON_MBWU must be shifted left by MPAMF_MBWUMON_IDR.SCALE bit positions to scale the value to the number of bytes.
- An optional capture register (MSMON_MBWU_CAPTURE) is loaded from the monitor register each time the selected capture event occurs. When a capture event occurs, the monitor register is copied to the capture register and the monitor register is optionally reset to zero.
- A Not-Ready (NRDY) bit (*Not-Ready Bit* on page 10-149) in the memory-bandwidth usage register (MSMON_MBWU) is set when the filter register or the control register is written. The NRDY bit is reset to 0 after a capture event. The NRDY bit is copied to the capture register along with the rest of the monitor register's content. This copy is made before the NRDY bit is reset. If the value of the NRDY bit in the capture register is 1, the captured resource usage should be viewed as representing an incomplete sampling interval. Therefore, the count should be assumed to be incorrect.

A capture event is needed if the optional capture register is implemented. The capture event causes the transfer of each monitor’s count register to its capture register and may optionally reset the count register.

If the count register is reset by a capture event, this allows reading (1) the bytes transferred that meet the criteria set in the filter and control registers during the interval between the last two capture events from MSMON_MBWU_CAPTURE, and (2) the bytes transferred that meet the criteria set in the filter and control registers since the last capture event from MSMON_MBWU.

Bandwidth usage can be computed in software from the count of bytes transferred as read from MSMON_MBWU or MSMON_MBWU_CAPTURE and the interval over which the count was collected.

There can be several sources of the capture event. The capture event source to use is specified in MSMON_CFG_MBWU_CTL.CAPT_EVNT (*Memory-mapped monitoring configuration registers* on page 11-210). It can be advantageous to use a single event to capture monitors in several MSCs simultaneously. A periodic capture event for multiple MSCs could be generated at the system level, perhaps using a generic timer, and distributed to the several MSCs.

The source of an external capture event is selected in MSMON_CFG_MBWU_CTL.CAPT_EVNT. A local capture event generator is present if MPAMF_MSMON_IDR.HAS_LOCAL_CAPT_EVNT == 1, and this generator generates events when certain values are written into MSMON_CAPT_EVNT.

If MSMON_CFG_MBWU_CTL.SCLEN == 0, the count is not scaled. If MSMON_CFG_MBWU_CTL.SCLEN == 1, the count in MSMON_MBWU is a scaled count.

The scaled count in MSMON_MBWU is the true count of bytes transferred, rounded to 2^SCALE and then shifted right by SCALE bit positions. The shift count, SCALE, is MPAMF_MBWUMON_IDR.SCALE.

SCALE is an implementation constant chosen for a monitoring point such that periodic sampling and reset of MSMON_MBWU_CAPTURE can count the highest traffic rates possible at the monitoring point without overflowing the VALUE field at a maximum sampling rate. The sampling rate is limited by the target use.

For example, if the maximum traffic that could pass the monitoring point is 300 GBps and the system environment supports capturing the counter 30 times per second, the counter must be scaled to no more than 2^31 - 1 counts per thirtieth of a second. This requires scaling the counter by a factor of at least 5, so the SCALE must be at least 3.

If the traffic to memory might be distributed across several MSCs (for example, across several memory channel controllers), a comprehensive measurement of bandwidth might require reading multiple memory-bandwidth usage monitors on those MSCs and summing the results. Capturing those monitors with the same system-level capture event allows correlated monitor values.

10.2.2 Cache-storage usage monitor

A cache-storage usage monitor is filtered by a PARTID and PMG. Each monitor has the following memory-mapped configuration registers:

- A filter register (MSMON_CFG_MBWU_FLT) that sets the PARTID and PMG to be monitored.
- A cache-storage usage register (MSON_CSU) that reports the amount of storage currently present within the cache allocated by the PARTID and PMG.
- A Not-Ready (NRDY) bit (*Not-Ready Bit* on page 10-149) in the cache-storage usage register (MSMON_CSU) that indicates that the value is not accurate. An implementation may set this bit if the value in the cache-storage usage register is not currently accurate, possibly because it is still being computed.
- An optional capture register (MSMON_CSU_CAPTURE) that is loaded from the cache-storage usage register each time the capture event occurs.

A capture event is needed if the optional capture register is implemented. The capture event causes the transfer of each monitor's cache-storage usage register to its optional capture register.

The source of the capture event is not specified here. It can be advantageous to use a single event to capture monitors in several MSCs simultaneously. A periodic capture event for multiple MSCs could be generated at the system level, perhaps using a generic timer, and distributed to the several MSCs.

The source of an external capture event is selected in MSMON_CFG_MBWU_CTL.CAPT_EVNT. A local capture event generator is present if MPAMF_MSMON_IDR.HAS_LOCAL_CAPT_EVNT == 1, and this generator generates events when certain values are written into MSMON_CAPT_EVNT.

If a monitor needs time to become accurate, the NRDY bit signals that the value is not yet accurate. Some methods of building cache-storage usage monitors may involve (1) a phase in which the monitor collects enough information to begin accurately tracking usage, or (2) a phase in which the measurement is kept accurate by tracking resource usage events. For example such a monitor might take tens of microseconds to complete the first phase before the value accurately tracks the actual resource usage. In this case, the NRDY bit would be kept at 1 until the monitor value becomes accurate.

The NRDY bit is included because some implementations may have timing restrictions between setting the filter register and reading the cache-storage usage register that may span thousands of PE cycles. Reading the monitor too soon is permitted to affect the accuracy of the readout, and it is indicated when the not-ready bit of the cache-storage usage register == 1.

10.3 Common features

All MPAM performance monitors have these features:

- Not-ready bit.
- Capture register.
- Overflow bit.

These features are described below.

10.3.1 Not-Ready Bit

The Not-Ready (NRDY) bit, in the MSMON_MBWU and MSMON_CSU registers, when set, indicates that the monitor does not have an accurate count or measurement yet, because the monitor's settings have been recently changed. If the monitor requires some time to establish a new count or measurement after its settings are changed, the NRDY bit must be set automatically when the settings are changed and reset when the count or measurement is accurately represented in the monitor.

In the absence of another change in settings, the NRDY bit must clear automatically within a maximum length of time. The maximum time that NRDY may be 1 is an implementation parameter that is discoverable in the firmware data value of MAX_NRDY_USEC for the MSC's monitor type.

Each instance of each type of monitor keeps its NRDY bit separately. For example, if MBWU monitor instance 3 is collecting memory bytes transferred for one partition and MBWU monitor instance 6 is later configured to collect for another partition, the configuration of MBWU monitor instance 6 must not disturb the on-going collection in MBWU monitor instance 3.

The NRDY bit of a monitor or capture register can be written to either state and may subsequently change state due to a capture event or a change in the configuration of the monitor.

If a monitor does not support the automatic behaviors of NRDY, this bit is permitted to be an RW bit with no additional functionality.

10.3.2 Capture event and capture register

A capture event causes every monitor that is configured to be sensitive to that event to be copied into that monitor's capture register.

Capture events may be local to the MSC or external to the MSC and may be software-initiated single events or a periodically repeating series of events. External capture events are system-defined. A generic counter can be used as the source of such an event, but this is not required. An external capture event could be distributed to all MSCs so that system-wide captures occur of all monitors sensitive to the external event. This permits using the various measurements for sums and differences because they measure the same period and (mostly) related resource usage.

A capture register for a monitor is loaded with the monitor's count or measurement and its NRDY bit when a capture event that is selected in the monitor's control register occurs. A capture event completes almost instantaneously, so no handshake is used for completion. However, the NRDY bit indicates whether a capture is not an accurate reading.

If the event is periodic, software can read the capture registers at any time to get the results captured when the most recent capture event occurred.

If it makes sense for the particular monitored value, the count or measurement can optionally be reset by the event. In this case, the value in the capture register represents a count over the capture-event period or a measurement over that period.

Local capture-event generator

If MPAMF_MSMON_IDR.HAS_LOCAL_CAPT_EVNT == 1, the MSMON_CAPT_EVNT register exists and generates capture events that are local to an MSC when it is written with a value that contains a 1 in the NOW bit position.

There are separate MSMON_CAPT_EVNT registers for Secure and Non-secure address spaces. The Non-secure version generates a local capture event to all Non-secure monitors within the MSC that have been configured to use MSMON_CFG_type_FLT.CAPT_EVNT == 7 (Table 10-1 on page 10-151). The Secure version of MSMON_CAPT_EVNT generates a local capture event to all Secure monitors within the MSC that have been configured to use CAPT_EVNT == 7 when MSMON_CAPT_EVNT is written with ALL == 0 and NOW == 1. When the ALL and NOW bits both == 1 in a write to Secure MSMON_CAPT_EVNT, the write generates a local capture event to all Secure and Non-secure monitors within the MSC that have been configured to use CAPT_EVNT == 7.

If MPAMF_MSMON_IDR.HAS_LOCAL_CAPT_EVNT == 0, local capture events are not generated and any monitors that have their control register set to CAPT_EVNT == 7 do not receive any capture events.

Reset on capture

Monitors that keep a count of events, or that accumulate counts such as bytes transferred, may be optionally reset after a capture event transfers the count to the monitor's capture register. This behavior on capture is controlled by the MSMON_CFG_*_CTL.CAPT_RESET bit. If CAPT_RESET == 1, the monitor count is reset to 0 immediately after the value is captured into the MSMON_*_CAPTURE register.

Monitors that report a current resource value, such as cache-storage usage, that cannot reasonably be reset, do not need to support reset on capture behavior. Arm recommends that these monitors have the CAPT_RESET bit as RAZ/WI.

10.3.3 Overflow bit

The MSMON_CFG_*_CTL.OFLOW_STATUS bit is set to 1 when the monitor counter overflows. This bit must be reset by writing 0 to the OFLOW_STATUS field.

The MSMON_CFG_*_CTL register contains fields to control MPAM behavior on an overflow. The OFLOW_FRZ bit, when set, freezes the counter after the count that caused it to overflow; when reset to 0, the counter continues to count after an overflow. The OFLOW_INTR bit, when set, signals an IMPLEMENTATION DEFINED interrupt when the counter overflows; when 0, no interrupt is signalled.

10.4 Monitor configuration

For each type of resource monitor, the number of monitor instances that are available is described in the corresponding MPAMF_<type>MON_IDR.NUM_MON field.

The MSMON_CFG_MON_SEL.MON_SEL field selects the monitor instance to configure. The MON_SEL monitor instance of monitor type, type, is accessed when an MSMON_CFG_<type> register is accessed.

All monitor types have two 32-bit configuration registers:

- MSMON_CFG_<type>_FLT (Table 10-1) has fields to select the PARTID and PMG to monitor.
- MSMON_CFG_<type>_CTL (Table 10-2) has controls for counting a subset of events, controlling overflow, and capture behavior.

Some monitor types may not require all fields, and fields not required must be RAZ/WI or RAO/WI.

Table 10-1 MSMON_CFG_<type>_FLT register template

Bits	Name	Description
15:0	PARTID	Configures the PARTID for the selected monitor to match. Matching of PARTID is enabled by MSMON_CFG_<type>_CTL.MATCH_PARTID.
23:16	PMG	Configures the PMG for the selected monitor to match. Matching of PMG is enabled by MSMON_CFG_<type>_CTL.MATCH_PMG.
31:24	Reserved	RAZ/WI.

Table 10-2 MSMON_CFG_<type>_CTL register template

Bits	Name		
7:0	TYPE	RO: Constant type indicating the type of the monitor. Currently assigned values are 0x42 for MBWU monitor, and 0x43 for CSU monitor. Other values less than 0x80 are reserved. Values greater than 0x80 are for use by IMPLEMENTATION DEFINED monitors.	
15:8	Reserved	RAZ/WI.	
16	MATCH_PARTID	**0**	Monitor events with any PARTID.
		1	Only monitor events with the PARTID matching MSMON_CFG_<type>_FLT.PARTID.
17	MATCH_PMG	**0**	Monitor events with any PMG.
		1	Only monitor events with the PMG matching MSMON_CFG_type_FLT.PMG.
19:18	Reserved	RAZ/WI	
23:20	SUBTYPE	A monitor can have other event-matching criteria. The meaning of values in this field can vary by monitor type. If not used by the monitor type, this field is RAZ/WI.	
24	OFLOW_FRZ	**0**	Monitor count wraps on overflow and continues to count.
		1	Monitor count freezes on overflow. The frozen value may be 0 or another value, if the monitor overflowed with an increment larger than 1.
25	OFLOW_INTR	**0**	No interrupt.
		1	On overflow, an implementation-specific interrupt is signaled.
26	OFLOW_STATUS	**1**	No overflow has occurred.
		1	At least one overflow has occurred since this bit was last written to 0.

Table 10-2 MSMON_CFG_<type>_CTL register template (continued)

Bits	Name	Description
27	CAPT_RESET	**0** Monitor is not reset on capture. **1** Monitor is reset on capture. If capture is not implemented for this monitor type, or the monitor is not a count that can be reasonably reset, this field is RAZ/WI.
30:28	CAPT_EVNT	Select the event that triggers capture from the following: **0** External capture event 1 (optional but recommended). **1** External capture event 2 (optional). **2** External capture event 2 (optional). **3** External capture event 3 (optional). **4** External capture event 4 (optional). **5** External capture event 5 (optional). **6** External capture event 6 (optional). **7** Capture occurs when the MSMON_CAPT_EVNT register is written. (optional). External capture events are system-defined. An external capture event could be distributed to many MSCs. The values marked as optional indicate capture-event sources that can be omitted in an implementation. Those values representing non-implemented event sources must not trigger a capture event. If capture is not implemented for the monitor, as indicated by MPAMF_<type>MON_IDR.HAS_CAPTURE == 0, this field is RAZ/WI.
31	EN	**0** The monitor is disabled and must not collect any information. **1** The monitor is enabled to collect information according to its configuration.

Chapter 11
Memory-Mapped Registers

This chapter contains the following sections:

11.1 Overview of MMRs

The MPAM behavior of an MSC is discovered and configured via memory-mapped registers (MMRs) in the MSC.

All MPAM MMRs are located on the MPAM feature page for the MSC (*MPAM feature page* on page 11-155). An MSC's MPAM feature page is located from information about the device, possibly provided via firmware data such as device tree or ACPI (Appendix B *MSC Firmware Data*).

An MPAM feature page exists in the Non-secure address space and another exists in the Secure address space. The addresses of the two MPAM feature pages of an MSC do not need to have the same base address. Arm recommends that the numerical base addresses of the Non-secure and Secure be sufficiently different that the numerical address ranges do not overlap.

MPAM MSC MMRs must support 32-bit access as a single access. There is no requirement that accesses of wider than 32 bits complete atomically.

There are MMRs for identifying MPAM parameters and options, the ID registers. These IDRs have the MPAMF prefix.

Other registers configure MPAM resource controls. These registers have the MPAMCFG prefix.

The resource monitor configuration and readout registers have the MSMON prefix.

Finally, there is a register to report the status of MPAM programming errors encountered in the MSC and a register to control MPAM interrupts.

11.1.1 Determining presence and location of MMRs

The MPAMF_IDR register is located at offset `0x0000` of the MPAM feature page. It indicates which MPAM resource controls are present in the MSC and the maximum PARTID and PMG supported in requests to the MSC. Other MPAMF ID registers are present if the corresponding MPAMF_IDR register bit is set and those registers identify the implemented values of architecturally-defined parameters associated with the particular class of MPAM resource control.

The MPAMF_IDR also indicates whether the MSC has MPAM monitors. If so, MPAMF_MSMON_IDR indicates which monitor types are supported by the MSC. Other monitor MPAMF ID registers are present if the corresponding bit in MPAMF_MSMON_IDR is set and those registers identify the implemented values of architecturally- defined parameters associated with the particular type of MPAM monitor.

The address of each MPAM MMR present in an MSC is located within the MPAM feature page for that component at a register-specific offset into that page. The offsets are given in tables in *Summary of memory-mapped registers* on page 11-159 and *MPAM feature page* on page 11-155.

11.1.2 Configuring resource controls for a partition

To configure the MPAM resource controls supported by an MSC for a PARTID:

1. Gain exclusive access to the MSC's partitioning configuration registers (for example, take a lock for the memory-mapped partitioning configuration registers, *Memory-mapped partitioning configuration registers* on page 11-189).
2. Write the PARTID to the component's MPAMCFG_PART_SEL.
3. Write to the MPAMCFG_* registers for the resource controls of the component.
4. Repeat step 3 to configure additional controls associated with the PARTID selected in step 2.
5. Repeat steps 2 through 4 to configure controls for additional PARTIDs.
6. Release exclusive access to the MSC's partitioning control configuration registers (for example, release the lock taken in step 1).

Repeat this procedure for each MSC.

The configuration registers are all the read-write registers that begin with MPAMCFG_*. That is all of the registers in *Memory-mapped partitioning configuration registers* on page 11-189. Before writing any of these registers, software must take a lock to prevent other software from accessing these registers until the lock is released. This is in part because the writing involves first putting a PARTID into the MPAMCFG_PART_SEL register and then writing a configuration value into one or more of the MPAM resource control's configuration registers (also MPAMCFG_* registers).

Software must also take a lock to read any MPAMCFG_* register, other than MPAMCFG_PART_SEL, because reading also involves first putting a PARTID into MPAMCFG_PART_SEL register and then reading a configuration value from one or more of the MPAMCFG_* registers.

There are two copies of MPAMCFG_PART_SEL, one for resource controls for the Secure PARTID space that are accessed from the Secure address space, and the other for resource controls for the Non-secure PARTID space that are accessed from the Non-secure address space. Because there are two copies, there can be separate locks for Secure MPAMCFG_PART_SEL and for Non-secure MPAMCFG_PART_SEL.

11.1.3 Configuring memory-system monitors

To configure the memory-system monitors supported by an MSC for a PARTID and PMG:

1. Gain exclusive access to the MSC's monitor configuration registers (for example, take a lock for the memory-mapped monitoring configuration registers, *Memory-mapped monitoring configuration registers* on page 11-210).
2. Write to the component's MSMON_CFG_MON_SEL to select one of the monitor instances available in the component.
3. Write to the MSMON_CFG_* registers for the instance of the monitor type.
4. Repeat step 3 to configure additional registers associated with the monitor instance.
5. Repeat steps 2 through 4 to configure additional monitor instances.
6. Release the exclusive access to the MSC's monitor configuration registers (for example, release the lock taken in step 1).

Repeat this procedure for each MSC.

Software must also take the lock to read any MSMON_* register, other than MSMON_CFG_MON_SEL, because reading involves first writing a monitor index into MSMON_CFG_MON_SEL and then reading an MSMON register.

The monitor configuration registers are all of the registers in *Memory-mapped monitoring configuration registers* on page 11-210. These registers have requirements similar to the MPAMCFG_* registers. The monitor configuration registers can have a separate lock or share the same lock as for the MPAMCFG_* registers. The selection register for monitors is MSMON_CFG_MON_SEL.

The configuration reading procedure of this section is also required to read the monitor and capture registers because these too are addressed by MSMON_CFG_MON_SEL.

There are two copies of MSMON_CFG_MON_SEL, one for Secure monitors that are accessed from the Secure address space and the other for Non-secure monitors that are accessed from the Non-secure address space. Because there are two copies, there can be separate locks for Secure MSMON_CFG_MON_SEL and for Non-secure MSMON_CFG_MON_SEL.

11.1.4 MPAM feature page

An MSC has an MPAM feature page in each of the secure and non-secure address spaces. An MPAM feature page is a block of addresses that contains all of the MPAM MSC MMRs in that address space. Each MPAM feature page base address must be aligned to a 4 KB boundary.

Each MPAM feature page must be completely contained within a single 64 KB aligned block so that it may be placed within a single 64KB page. Non-MPAM MMRs of the MSC are permitted within the 64 KB block if those MMRs are also to be trapped to a hypervisor.

Secure and Non-secure address space

If the MSC supports the Secure address space (NS == 0), the Secure MPAM feature page must exist. The Non-secure MPAM feature page must always exist.

MMRs describing (IDRs) or controlling (MPAMCFG*) Secure PARTIDs are within the Secure MPAM feature page, and those describing or controlling Non-secure PARTIDs are within the Non-secure MPAM feature page.

MPAM MMRs only in the Secure address space

Certain MPAM MMRs are only present within the MPAM feature page when accessed via the Secure address space (NS = 0). MPAMF_SIDR is the only MMR accessible only via the Secure address space.

Read-only MPAM MMRs permitted to read the same or differently

Some of the read-only MPAM MMRs are permitted to have the same or different contents between the Secure and Non-secure MPAM feature pages This includes all of the MPAMF*IDR registers. If the information regarding Secure and Non-secure PARTIDs is the same in an MPAMF*IDR, then the register is permitted to have the same contents.

These registers are permitted to be shared if the same or banked if different in the two address spaces:

MPAMF_IDR	MPAMF_IMPL_IDR	MPAMF_CPOR_IDR
MPAMF_CCAP_IDR	MPAMF_MBW_IDR	MPAMF_PRI_IDR
MPAMF_PARTID_NRW_IDR	MPAMF_MSMON_IDR	MPAMF_CSUMON_IDR
MPAMF_MBWUMON_IDR		

MPAM MMRs that must have the same contents

Two registers must have the same contents between the Secure and Non-secure MPAM feature pages. These registers contain read-only values that must read as the same value in the two address spaces:

MPAMF_IIDR	MPAMF_AIDR

MPAM MMRs that must be separate registers for each address space

Most MPAM MMRs, such as the following, must be separate and have Secure and Non-secure versions that are accessed via the corresponding Secure and Non-secure MPAM feature pages:

MPAMF_ECR	MPAMCFG_PART_SEL	MSMON_CFG_MON_SEL
MPAMF_ESR	MPAMCFG_MBW_MAX	MSMON_CFG_CSU_CTL
	MPAMCFG_MBW_MIN	MSMON_CFG_CSU_FLT
MPAMCFG_CMAX	MPAMCFG_MBW_PBM	MSMON_CSU
MPAMCFG_CPBM	MPAMCFG_MBW_PROP	MSMON_CSU_CAPTURE
	MPAMCFG_MBW_WINWD	MSMON_CFG_MBWU_CTL
MPAMCFG_PRI		MSMON_CFG_MBWU_FLT
MPAMCFG_INTPARTID		MSMON_MBWU
		MSMON_MBWU_CAPTURE

Accesses to locations where there is no register in the address space of the access

Access to MPAM MMR address where there is no register in the address space of the access must be treated as reserved MPAM feature page locations according to *IMPLEMENTATION-DEFINED MPAM memory-mapped registers and reserved MPAM feature page locations* on page 11-157, except for the MPAMCFG_MBW_PBM and MPAMCFG_CPBM as described in *Permitted truncation of an MPAM feature page* on page 11-157.

Permitted truncation of an MPAM feature page

An MPAM feature page may be shortened in just two cases:

- If MPAMCFG_MBW_PBM is not implemented (MPAMF_IDR.HAS_MBW_PART == 0' || (MPAM_IDR.HAS_MBW_PART == 1 && MPAM_MBW_IDR.HAS_PBM == 0)), the maximum offset for the MPAM feature page is `0x01FFF`.
- If MPAMCFG_MBW_PBM is not implemented and MPAMCFG_CPBM is not implemented (MPAMF_IDR.HAS_CPOR == 0), the maximum offset for the MPAM feature page is `0x00FFF`.

11.1.5 Minimum required MPAM memory-mapped registers

If an MSC has any support for MPAM, the following registers are required:

- MPAMF_IDR.
- MPAMF_AIDR.
- MPAMF_IIDR.
- MPAMF_SIDR, if the Secure address space is supported.

If an MSC supports any resource controls, the following registers are also required:

- MPAMCFG_PART_SEL.

If an MSC supports any resource monitors, the following registers are also required:

- MPAMF_MSMON_IDR.
- MSMON_CFG_MON_SEL.

If an MSC can detect any errors, it must implement:

- MPAMF_ESR.
- MPAMF_ECR.

MSC MPAM MMRs not mentioned in this section are optional and expected to be implemented only when the resource control or monitor that the register supports is implemented.

11.1.6 IMPLEMENTATION-DEFINED MPAM memory-mapped registers and reserved MPAM feature page locations

IMPLEMENTATION DEFINED MPAM memory mapped registers are permitted in the MPAM feature page at offsets equal to or greater than `0x3000`.

All locations in the MPAM feature page at offsets less than the maximum MPAM feature page offset defined in *Permitted truncation of an MPAM feature page* are reserved to the architecture. Within that address range:

- Reads and writes of unallocated locations are reserved accesses.
- Reads and writes of locations for registers that are not implemented are reserved accesses, including register locations for:
  - OPTIONAL MPAM MSC features that are not implemented.
  - ID registers for OPTIONAL MPAM MSC features that are not implemented and indicated as not implemented in ID registers that are implemented.
- Locations that are beyond the implemented width of a register as given in the corresponding ID register but within the range of locations allocated by the architecture are reserved accesses.
- Reads of WO locations are reserved accesses.
- Writes to RO locations are reserved accesses.

The architecture requires reserved accesses to be implemented as RAZ/WI. However, software must not rely on this property as the behavior of reserved values might change in a future revision of the MPAM Extension architecture. Software must treat reserved accesses as RES0.

11.2 Summary of memory-mapped registers

Table 11-1 lists the external MPAM registers in order of register offset.

Table 11-1 Index of exterenal MPAM registers ordered by offset

Register	Offset	Length	Description, see
MPAMF_IDR	0x0000	32	*MPAMF_IDR, MPAM Features Identification Register* on page 11-170
MPAMF_SIDR	0x0008	32	*MPAMF_SIDR, MPAM Features Secure Identification Register* on page 11-188
MPAMF_IIDR	0x0018	32	*MPAMF_IIDR, MPAM Implementation Identification Register* on page 11-173
MPAMF_AIDR	0x0020	32	*MPAMF_AIDR, MPAM Architecture Identification Register* on page 11-162
MPAMF_IMPL_IDR	0x0028	32	*MPAMF_IMPL_IDR, MPAM Implementation-Specific Partitioning Feature Identification Register* on page 11-175
MPAMF_CPOR_IDR	0x0030	32	*MPAMF_CPOR_IDR, MPAM Features Cache Portion Partitioning ID register* on page 11-166
MPAMF_CCAP_IDR	0x0038	32	*MPAMF_CCAP_IDR, MPAM Features Cache Capacity Partitioning ID register* on page 11-164
MPAMF_MBW_IDR	0x0040	32	*MPAMF_MBW_IDR, MPAM Memory Bandwidth Partitioning Identification Register* on page 11-177
MPAMF_PRI_IDR	0x0048	32	*MPAMF_PRI_IDR, MPAM Priority Partitioning Identification Register* on page 11-186
MPAMF_PARTID_NRW_IDR	0x0050	32	*MPAMF_PARTID_NRW_IDR, MPAM PARTID Narrowing ID register* on page 11-184
MPAMF_MSMON_IDR	0x0080	32	*MPAMF_MSMON_IDR, MPAM Resource Monitoring Identification Register* on page 11-182
MPAMF_CSUMON_IDR	0x0088	32	*MPAMF_CSUMON_IDR, MPAM Features Cache Storage Usage Monitoring ID register* on page 11-168
MPAMF_MBWUMON_IDR	0x0090	32	*MPAMF_MBWUMON_IDR, MPAM Features Memory Bandwidth Usage Monitoring ID register* on page 11-180
MPAMF_ECR	0x00F0	32	*MPAMF_ECR, MPAM Error Control Register* on page 11-236
MPAMF_ESR	0x00F8	32	*MPAMF_ESR, MPAM Error Status Register* on page 11-238
MPAMCFG_PART_SEL	0x0100	32	*MPAMCFG_PART_SEL, MPAM Partition Configuration Selection Register* on page 11-206
MPAMCFG_CMAX	0x0108	32	*MPAMCFG_CMAX, MPAM Cache Maximum Capacity Partition Configuration Register* on page 11-190
MPAMCFG_MBW_MIN	0x0200	32	*MPAMCFG_MBW_MIN, MPAM Cache Maximum Capacity Partition Configuration Register* on page 11-198
MPAMCFG_MBW_MAX	0x0208	32	*MPAMCFG_MBW_MAX, MPAM Memory Bandwidth Maximum Partition Configuration Register* on page 11-196
MPAMCFG_MBW_WINWD	0x0220	32	*MPAMCFG_MBW_WINWD, MPAM Memory Bandwidth Partitioning Window Width Configuration Register* on page 11-204

Table 11-1 Index of exterenal MPAM registers ordered by offset (continued)

Register	Offset	Length	Description, see
MPAMCFG_PRI	0x0400	32	*MPAMCFG_PRI, MPAM Priority Partition Configuration Register* on page 11-208
MPAMCFG_MBW_PROP	0x0500	32	*MPAMCFG_MBW_PROP, MPAM Memory Bandwidth Proportional Stride Partition Configuration Register* on page 11-202
MPAMCFG_INTPARTID	0x0600	32	*MPAMCFG_INTPARTID, MPAM Internal PARTID Narrowing Configuration Register* on page 11-194
MSMON_CFG_MON_SEL	0x0800	32	*MSMON_CFG_MON_SEL, MPAM Monitor Instance Selection Register* on page 11-225
MSMON_CAPT_EVNT	0x0808	32	*MSMON_CAPT_EVNT, MPAM Capture Event Generation Register* on page 11-211
MSMON_CFG_CSU_FLT	0x0810	32	*MSMON_CFG_CSU_FLT, MPAM Memory System Monitor Configure Cache Storage Usage Monitor Filter Register* on page 11-217
MSMON_CFG_CSU_CTL	0x0818	32	*MSMON_CFG_CSU_CTL, MPAM Memory System Monitor Configure Cache Storage Usage Monitor Control Register* on page 11-213
MSMON_CFG_MBWU_FLT	0x0820	32	*MSMON_CFG_MBWU_FLT, MPAM Memory System Monitor Configure Memory Bandwidth Usage Monitor Filter Register* on page 11-223
MSMON_CFG_MBWU_CTL	0x0828	32	*MSMON_CFG_MBWU_CTL, MPAM Memory System Monitor Configure Memory Bandwidth Usage Monitor Control Register* on page 11-219
MSMON_CSU	0x0840	32	*MSMON_CSU, MPAM Cache Storage Usage Monitor Register* on page 11-227
MSMON_CSU_CAPTURE	0x0848	32	*MSMON_CSU_CAPTURE, MPAM Cache Storage Usage Monitor Capture Register* on page 11-229
MSMON_MBWU	0x0860	32	*MSMON_MBWU, MPAM Memory Bandwidth Usage Monitor Register* on page 11-231
MSMON_MBWU_CAPTURE	0x0868	32	*MSMON_MBWU_CAPTURE, MPAM Memory Bandwidth Usage Monitor Capture Register* on page 11-233
MPAMCFG_CPBM	0x1000	32768	*MPAMCFG_CPBM, MPAM Cache Portion Bitmap Partition Configuration Register* on page 11-192
MPAMCFG_MBW_PBM	0x2000	4096	*MPAMCFG_MBW_PBM, MPAM Bandwidth Portion Bitmap Partition Configuration Register* on page 11-200

11.3 Memory-mapped ID register description

This section lists the external ID registers.

11.3.1 MPAMF_AIDR, MPAM Architecture Identification Register

The MPAMF_AIDR characteristics are:

Purpose

The MPAMF_AIDR is a 32-bit read-only register that identifies the version of the MPAM architecture that this MSC implements.

Note: The following values are defined for bits [7:0]:

- `0x10` == MPAM architecture v1.0

Configurations

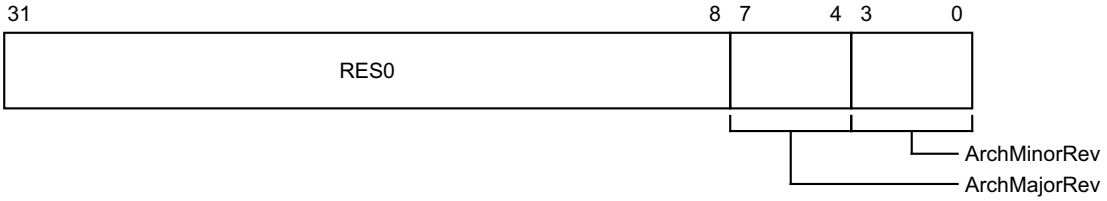
The power domain of MPAMF_AIDR is IMPLEMENTATION DEFINED.

Attributes

MPAMF_AIDR is a 32-bit register.

Field descriptions

The MPAMF_AIDR bit assignments are:



Bits [31:8]

Reserved, RES0.

ArchMajorRev, bits [7:4]

Major revision of the MPAM architecture implemented by the MSC.

ArchMinorRev, bits [3:0]

Minor revision of the MPAM architecture implemented by the MSC.

Accessing the MPAMF_AIDR:

This register is within the MPAM feature page memory frames. In a system that supports Secure and Non-secure memory maps, there must be both Secure and Non-secure MPAM feature pages.

MPAMF_AIDR is read-only.

MPAMF_AIDR must be readable from the Non-secure and Secure MPAM feature pages.

MPAMF_AIDR must have the same contents in the Secure and Non-secure MPAM feature pages.

MPAMF_AIDR can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_s	`0x0020`	MPAMF_AIDR

This interface is accessible as follows:

- Accesses to this register are RO.

MPAMF_AIDR can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_ns	0x0020	MPAMF_AIDR

This interface is accessible as follows:

- Accesses to this register are RO.

11.3.2 MPAMF_CCAP_IDR, MPAM Features Cache Capacity Partitioning ID register

The MPAMF_CCAP_IDR characteristics are:

Purpose

The MPAMF_CCAP_IDR is a 32-bit read-only register that indicates the number of fractional bits in MPAMCFG_CMAX for this MSC. MPAMF_CCAP_IDR_s indicates the number of fractional bits in the Secure instance of MPAMCFG_CMAX. MPAMF_CCAP_IDR_ns indicates the number of fractional bits in the Non-secure instance of MPAMCFG_CMAX.

Configurations

The power domain of MPAMF_CCAP_IDR is IMPLEMENTATION DEFINED.

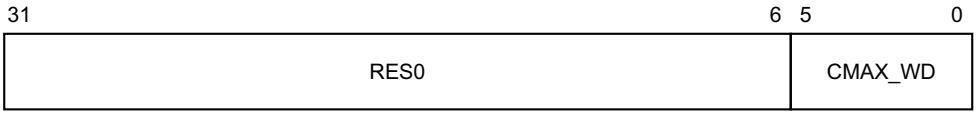
This register is present only when MPAMF_IDR.HAS_CCAP_PART == 1. Otherwise, direct accesses to MPAMF_CCAP_IDR are RES0.

Attributes

MPAMF_CCAP_IDR is a 32-bit register.

Field descriptions

The MPAMF_CCAP_IDR bit assignments are:



Bits [31:6]

Reserved, RES0.

CMAX_WD, bits [5:0]

Number of fractional bits implemented in the cache capacity partitioning control, MPAMCFG_CMAX.CMAX, of this device. See MPAMCFG_CMAX.

This field must contain a value from 1 to 16, inclusive.

Accessing the MPAMF_CCAP_IDR:

This register is within the MPAM feature page memory frames. In a system that supports Secure and Non-secure memory maps, there must be both Secure and Non-secure MPAM feature pages.

MPAMF_CCAP_IDR is read-only.

MPAMF_CCAP_IDR must be readable from the Non-secure and Secure MPAM feature pages.

MPAMF_CCAP_IDR is permitted to have the same contents when read from either the Secure and Non-secure MPAM feature pages unless the register contents is different for Secure and Non-secure versions, when there must be separate registers in the Secure (MPAMF_CCAP_IDR_s) and Non-secure (MPAMF_CCAP_IDR_ns) MPAM feature pages.

MPAMF_CCAP_IDR can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_s	0x0038	MPAMF_CCAP_IDR_s

This interface is accessible as follows:

- Accesses to this register are RO.

MPAMF_CCAP_IDR can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_ns	0x0038	MPAMF_CCAP_IDR_ns

This interface is accessible as follows:

- Accesses to this register are RO.

11.3.3 MPAMF_CPOR_IDR, MPAM Features Cache Portion Partitioning ID register

The MPAMF_CPOR_IDR characteristics are:

Purpose

The MPAMF_CPOR_IDR is a 32-bit read-only register that indicates the number of bits in MPAMCFG_CPBM for this MSC. MPAMF_CPOR_IDR_s indicates the number of bits in the Secure instance of MPAMCFG_CPBM. MPAMF_CPOR_IDR_ns indicates the number of bits in the Non-secure instance of MPAMCFG_CPBM.

Configurations

The power domain of MPAMF_CPOR_IDR is IMPLEMENTATION DEFINED.

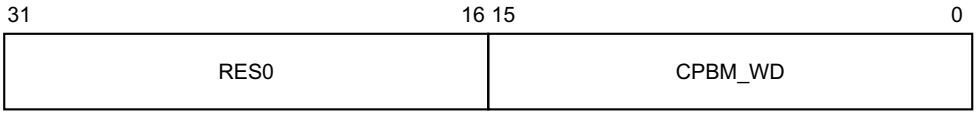
This register is present only when MPAMF_IDR.HAS_CPOR == 1. Otherwise, direct accesses to MPAMF_CPOR_IDR are RES0.

Attributes

MPAMF_CPOR_IDR is a 32-bit register.

Field descriptions

The MPAMF_CPOR_IDR bit assignments are:



Bits [31:16]

Reserved, RES0.

CPBM_WD, bits [15:0]

Number of bits in the cache portion partitioning bit map of this device. See MPAMCFG_CPBM.

This field must contain a value from 1 to 32768, inclusive. Values greater than 32 require a group of 32-bit registers to access the CPBM, up to 1024 if CPBM_WD is the largest value.

Accessing the MPAMF_CPOR_IDR:

This register is within the MPAM feature page memory frames. In a system that supports Secure and Non-secure memory maps, there must be both Secure and Non-secure MPAM feature pages.

MPAMF_CPOR_IDR is read-only.

MPAMF_CPOR_IDR must be readable from the Non-secure and Secure MPAM feature pages.

MPAMF_CPOR_IDR is permitted to have the same contents when read from either the Secure and Non-secure MPAM feature pages unless the register contents is different for Secure and Non-secure versions, when there must be separate registers in the Secure (MPAMF_CPOR_IDR_s) and Non-secure (MPAMF_CPOR_IDR_ns) MPAM feature pages.

MPAMF_CPOR_IDR can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_s	0x0030	MPAMF_CPOR_IDR_s

This interface is accessible as follows:

- Accesses to this register are RO.

MPAMF_CPOR_IDR can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_ns	0x0030	MPAMF_CPOR_IDR_ns

This interface is accessible as follows:

- Accesses to this register are RO.

11.3.4 MPAMF_CSUMON_IDR, MPAM Features Cache Storage Usage Monitoring ID register

The MPAMF_CSUMON_IDR characteristics are:

Purpose

The MPAMF_CSUMON_IDR is a 32-bit read-only register that indicates the number of cache storage usage monitors for this MSC. MPAMF_CSUMON_IDR_s indicates the number of Secure cache storage usage monitors. MPAMF_CSUMON_IDR_ns indicates the number of Non-secure cache storage usage monitors.

Configurations

The power domain of MPAMF_CSUMON_IDR is IMPLEMENTATION DEFINED.

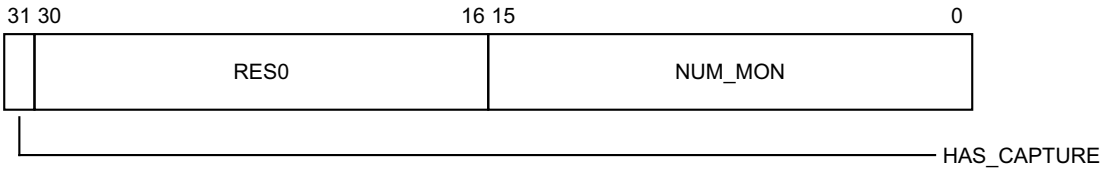
This register is present only when MPAMF_IDR.HAS_MSMON == 1 and MPAMF_MSMON_IDR.MSMON_CSU == 1. Otherwise, direct accesses to MPAMF_CSUMON_IDR are RES0.

Attributes

MPAMF_CSUMON_IDR is a 32-bit register.

Field descriptions

The MPAMF_CSUMON_IDR bit assignments are:



HAS_CAPTURE, bit [31]

The MSC's implementation supports copying an MSMON_CSU to the corresponding MSMON_CSU_CAPTURE on a capture event.

`0b0` MSMON_CSU_CAPTURE is not implemented and there is no support for capture events in this component's CSU monitor.

`0b1` The MSMON_CSU_CAPTURE register is implemented and this component's CSU monitor supports the capture event behavior.

Bits [30:16]

Reserved, RES0.

NUM_MON, bits [15:0]

The number of cache storage usage monitors implemented in this MSC.

Accessing the MPAMF_CSUMON_IDR:

This register is within the MPAM feature page memory frames. In a system that supports Secure and Non-secure memory maps, there must be both Secure and Non-secure MPAM feature pages.

MPAMF_CSUMON_IDR is read-only.

MPAMF_CSUMON_IDR must be readable from the Non-secure and Secure MPAM feature pages.

MPAMF_CSUMON_IDR is permitted to have the same contents when read from either the Secure and Non-secure MPAM feature pages unless the register contents is different for Secure and Non-secure versions, when there must be separate registers in the Secure (MPAMF_CSUMON_IDR_s) and Non-secure (MPAMF_CSUMON_IDR_ns) MPAM feature pages.

MPAMF_CSUMON_IDR can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_s	0x0088	MPAMF_CSUMON_IDR_s

This interface is accessible as follows:

- Accesses to this register are RO.

MPAMF_CSUMON_IDR can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_ns	0x0088	MPAMF_CSUMON_IDR_ns

This interface is accessible as follows:

- Accesses to this register are RO.

11.3.5 MPAMF_IDR, MPAM Features Identification Register

The MPAMF_IDR characteristics are:

Purpose

The MPAMF_IDR is a 32-bit read-only register that indicates which memory partitioning and monitoring features are present on this MSC. MPAMF_IDR_s indicates the MPAM features accessed from the Secure MPAM feature page. MPAMF_IDR_ns indicates the MPAM features accessed from the Non-secure MPAM feature page.

Configurations

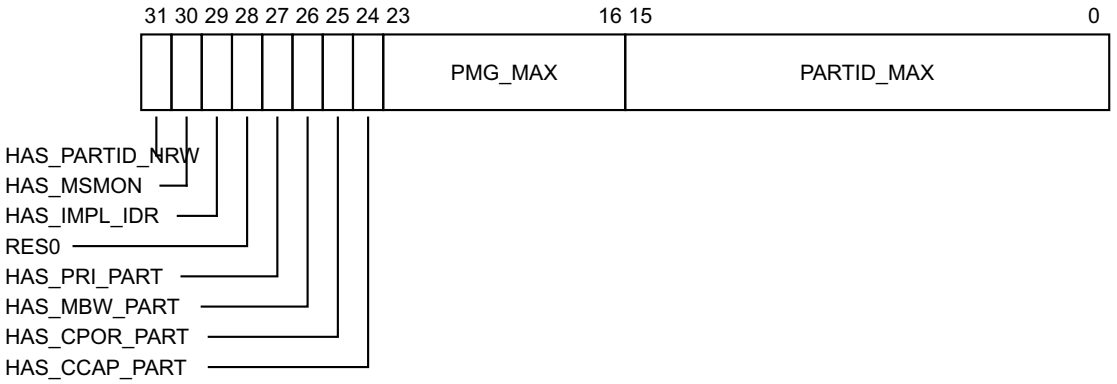
The power domain of MPAMF_IDR is IMPLEMENTATION DEFINED.

Attributes

MPAMF_IDR is a 32-bit register.

Field descriptions

The MPAMF_IDR bit assignments are:



HAS_PARTID_NRW, bit [31]

Has PARTID narrowing.

`0b0` Does not have MPAMF_PARTID_NRW_IDR, MPAMCFG_INTPARTID or intPARTID mapping support.

`0b1` Supports the MPAMF_PARTID_NRW_IDR, MPAMCFG_INTPARTID registers.

HAS_MSMON, bit [30]

Has resource monitors. Indicates whether this MSC has MPAM resource monitors.

`0b0` Does not support MPAM resource monitoring by groups or MPAMF_MSMON_IDR.

`0b1` Supports resource monitoring by matching a combination of PARTID and PMG. See MPAMF_MSMON_IDR.

HAS_IMPL_IDR, bit [29]

Has MPAMF_IMPL_IDR. Indicates whether this MSC has the implementation-specific MPAM features register, MPAMF_IMPL_IDR.

`0b0` Does not have MPAMF_IMPL_IDR.

`0b1` Has MPAMF_IMPL_IDR.

Bit [28]

Reserved, RES0.

HAS_PRI_PART, bit [27]

Has priority partitioning. Indicates whether this MSC implements MPAM priority partitioning and MPAMF_PRI_IDR.

`0b0` Does not support priority partitioning or have MPAMF_PRI_IDR.

`0b1` Has MPAMF_PRI_IDR.

HAS_MBW_PART, bit [26]

Has memory bandwidth partitioning. Indicates whether this MSC implements MPAM memory bandwidth partitioning and MPAMF_MBW_IDR.

`0b0` Does not support memory bandwidth partitioning or have MPAMF_MBW_IDR register.

`0b1` Has MPAMF_MBW_IDR register.

HAS_CPOR_PART, bit [25]

Has cache portion partitioning. Indicates whether this MSC implements MPAM cache portion partitioning and MPAMF_CPOR_IDR.

`0b0` Does not support cache portion partitioning or have MPAMF_CPOR_IDR or MPAMCFG_CPBM registers.

`0b1` Has MPAMF_CPOR_IDR and MPAMCFG_CPBM registers.

HAS_CCAP_PART, bit [24]

Has cache capacity partitioning. Indicates whether this MSC implements MPAM cache capacity partitioning and the MPAMF_CCAP_IDR and MPAMCFG_CMAX registers.

`0b0` Does not support cache capacity partitioning or have MPAMF_CCAP_IDR and MPAMCFG_CMAX registers.

`0b1` Has MPAMF_CCAP_IDR and MPAMCFG_CMAX registers.

PMG_MAX, bits [23:16]

Maximum value of Non-secure PMG supported by this component.

PARTID_MAX, bits [15:0]

Maximum value of Non-secure PARTID supported by this component.

Accessing the MPAMF_IDR:

This register is within the MPAM feature page memory frames. In a system that supports Secure and Non-secure memory maps, there must be both Secure and Non-secure MPAM feature pages.

MPAMF_IDR is read-only.

MPAMF_IDR must be readable from the Non-secure and Secure MPAM feature pages.

MPAMF_IDR is permitted to have the same contents when read from either the Secure and Non-secure MPAM feature pages unless the register contents is different for Secure and Non-secure versions, when there must be separate registers in the Secure (MPAMF_IDR_s) and Non-secure (MPAMF_IDR_ns) MPAM feature pages.

MPAMF_IDR can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_s	`0x0000`	MPAMF_IDR_s

This interface is accessible as follows:

- Accesses to this register are RO.

MPAMF_IDR can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_ns	0x0000	MPAMF_IDR_ns

This interface is accessible as follows:

- Accesses to this register are RO.

11.3.6 MPAMF_IIDR, MPAM Implementation Identification Register

The MPAMF_IIDR characteristics are:

Purpose

The MPAMF_IIDR is a 32-bit read-only register that gives identification information to uniquely define the MSC.

Configurations

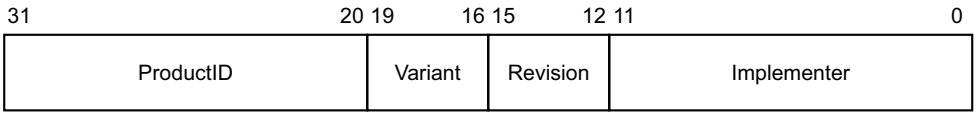
The power domain of MPAMF_IIDR is IMPLEMENTATION DEFINED.

Attributes

MPAMF_IIDR is a 32-bit register.

Field descriptions

The MPAMF_IIDR bit assignments are:



ProductID, bits [31:20]

IMPLEMENTATION DEFINED.

IMPLEMENTATION DEFINED value identifying the MPAM MSC.

Variant, bits [19:16]

IMPLEMENTATION DEFINED.

IMPLEMENTATION DEFINED value used to distinguish product variants, or major revisions of the product.

Revision, bits [15:12]

IMPLEMENTATION DEFINED.

IMPLEMENTATION DEFINED value used to distinguish minor revisions of the product.

Implementer, bits [11:0]

Contains the JEP106 code of the company that implemented the MPAM MSC.

[11:8] must contain the JEP106 continuation code of the implementer.

[7] must always be 0.

[6:0] must contain the JEP106 identity code of the implementer.

For an Arm implementation, bits[11:0] are `0x43B`.

Accessing the MPAMF_IIDR:

This register is within the MPAM feature page memory frames. In a system that supports Secure and Non-secure memory maps, there must be both Secure and Non-secure MPAM feature pages.

MPAMF_IIDR is read-only.

MPAMF_IIDR must be readable from the Non-secure and Secure MPAM feature pages.

MPAMF_IIDR must have the same contents in the Secure and Non-secure MPAM feature pages.

MPAMF_IIDR can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_s	0x0018	MPAMF_IIDR

This interface is accessible as follows:

- Accesses to this register are RO.

MPAMF_IIDR can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_ns	0x0018	MPAMF_IIDR

This interface is accessible as follows:

- Accesses to this register are RO.

11.3.7 MPAMF_IMPL_IDR, MPAM Implementation-Specific Partitioning Feature Identification Register

The MPAMF_IMPL_IDR characteristics are:

Purpose

The MPAMF_IMPL_IDR is a 32-bit read-only register that indicates the implementation-defined partitioning and monitoring features and parameters of the MSC. MPAMF_IMPL_IDR_s indicates implementation-defined partitioning and monitoring features accessed from the Secure MPAM feature page. MPAMF_IMPL_IDR_ns indicates those accessed from the Non-secure MPAM feature page.

Configurations

The power domain of MPAMF_IMPL_IDR is IMPLEMENTATION DEFINED.

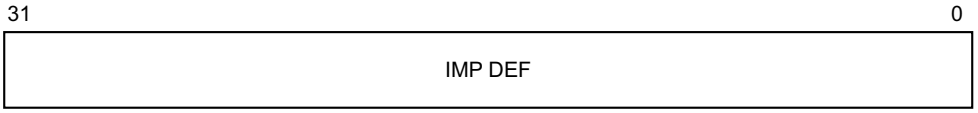
This register is present only when MPAMF_IDR.HAS_IMPL_IDR == 1. Otherwise, direct accesses to MPAMF_IMPL_IDR are RES0.

Attributes

MPAMF_IMPL_IDR is a 32-bit register.

Field descriptions

The MPAMF_IMPL_IDR bit assignments are:



IMP DEF, bits [31:0]

IMPLEMENTATION DEFINED.

All 32 bits of this register are available to be used as the implementer sees fit to indicate the presence of implementation-defined MPAM features in this MSC and to give additional implementation-specific read-only information about the parameters of implementation-specific MPAM features to software.

Accessing the MPAMF_IMPL_IDR:

This register is within the MPAM feature page memory frames. In a system that supports Secure and Non-secure memory maps, there must be both Secure and Non-secure MPAM feature pages.

MPAMF_IMPL_IDR is read-only.

MPAMF_IMPL_IDR must be readable from the Non-secure and Secure MPAM feature pages.

MPAMF_IMPL_IDR is permitted to have the same contents when read from either the Secure and Non-secure MPAM feature pages unless the register contents is different for Secure and Non-secure versions, when there must be separate registers in the Secure (MPAMF_IMPL_IDR_s) and Non-secure (MPAMF_IMPL_IDR_ns) MPAM feature pages.

MPAMF_IMPL_IDR can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_s	0x0028	MPAMF_IMPL_IDR_s

This interface is accessible as follows:

- Accesses to this register are RO.

MPAMF_IMPL_IDR can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_ns	0x0028	MPAMF_IMPL_IDR_ns

This interface is accessible as follows:

- Accesses to this register are RO.

11.3.8 MPAMF_MBW_IDR, MPAM Memory Bandwidth Partitioning Identification Register

The MPAMF_MBW_IDR characteristics are:

Purpose

The MPAMF_MBW_IDR is a 32-bit read-only register that indicates which MPAM bandwidth partitioning features are present on this MSC. MPAMF_MBW_IDR_s indicates bandwidth partitioning features accessed from the Secure MPAM feature page. MPAMF_MBW_IDR_ns indicates bandwidth partitioning features accessed from the Non-secure MPAM feature page.

Configurations

The power domain of MPAMF_MBW_IDR is IMPLEMENTATION DEFINED.

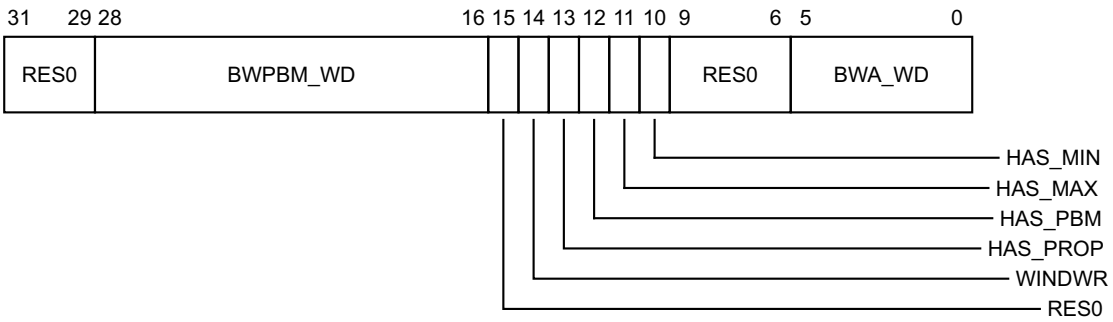
This register is present only when MPAMF_IDR.HAS_MBW_PART == 1. Otherwise, direct accesses to MPAMF_MBW_IDR are RES0.

Attributes

MPAMF_MBW_IDR is a 32-bit register.

Field descriptions

The MPAMF_MBW_IDR bit assignments are:



Bits [31:29]

Reserved, RES0.

BWPBM_WD, bits [28:16]

Bandwidth portion bitmap width.

The number of bandwidth portion bits in MPAMCFG_MBW_PBM.BWPBM.

This field must contain a value from 1 to 4096, inclusive. Values greater than 32 require a group of 32-bit registers to access the BWPBM, up to 128 if BWPBM_WD is the largest value.

Bit [15]

Reserved, RES0.

WINDWR, bit [14]

Indicates the bandwidth accounting period register is writable.

`0b0` The bandwidth accounting period is readable from MPAMCFG_MBW_WINWD which might be fixed or vary due to clock rate reconfiguration of the memory channel or memory controller.

`0b1` The bandwidth accounting width is readable and writable per partition in MPAMCFG_MBW_WINWD.

HAS_PROP, bit [13]

Indicates that this MSC implements proportional stride bandwidth partitioning and the MPAMCFG_MBW_PROP register.

`0b0` There is no memory bandwidth proportional stride control and no MPAMCFG_MBW_PROP register.

`0b1` The MPAMCFG_MBW_PROP register exists and the proportional stride memory bandwidth allocation scheme is supported.

HAS_PBM, bit [12]

Indicates that this MSC implements bandwidth portion partitioning and the MPAMCFG_MBW_PBM register.

`0b0` There is no memory bandwidth portion control and no MPAMCFG_MBW_PBM register.

`0b1` The MPAMCFG_MBW_PBM register exists and the memory bandwidth portion allocation scheme exists.

HAS_MAX, bit [11]

Indicates that this MSC implements maximum bandwidth partitioning and the MPAMCFG_MBW_MAX register.

`0b0` There is no maximum memory bandwidth control and no MPAMCFG_MBW_MAX register.

`0b1` The MPAMCFG_MBW_MAX register exists and the maximum memory bandwidth allocation scheme is supported.

HAS_MIN, bit [10]

Indicates that this MSC implements minimum bandwidth partitioning.

`0b0` There is no minimum memory bandwidth control and no MPAMCFG_MBW_MIN register.

`0b1` The MPAMCFG_MBW_MIN register exists and the minimum memory bandwidth allocation scheme is supported.

Bits [9:6]

Reserved, RES0.

BWA_WD, bits [5:0]

Number of implemented bits in the bandwidth allocation fields: MIN, MAX and STRIDE. See MPAMCFG_MBW_MIN, MPAMCFG_MBW_MAX and MPAMCFG_MBW_PROP.

This field must have a value from 1 to 16, inclusive.

Accessing the MPAMF_MBW_IDR:

This register is within the MPAM feature page memory frames. In a system that supports Secure and Non-secure memory maps, there must be both Secure and Non-secure MPAM feature pages.

MPAMF_MBW_IDR is read-only.

MPAMF_MBW_IDR must be readable from the Non-secure and Secure MPAM feature pages.

MPAMF_MBW_IDR is permitted to have the same contents when read from either the Secure and Non-secure MPAM feature pages unless the register contents is different for Secure and Non-secure versions, when there must be separate registers in the Secure (MPAMF_MBW_IDR_s) and Non-secure (MPAMF_MBW_IDR_ns) MPAM feature pages.

MPAMF_MBW_IDR can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_s	0x0040	MPAMF_MBW_IDR_s

This interface is accessible as follows:

- Accesses to this register are RO.

MPAMF_MBW_IDR can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_ns	0x0040	MPAMF_MBW_IDR_ns

This interface is accessible as follows:

- Accesses to this register are RO.

11.3.9 MPAMF_MBWUMON_IDR, MPAM Features Memory Bandwidth Usage Monitoring ID register

The MPAMF_MBWUMON_IDR characteristics are:

Purpose

The MPAMF_MBWUMON_IDR is a 32-bit read-only register that indicates the number of memory bandwidth usage monitors for this MSC. MPAMF_MBWUMON_IDR_s indicates the number of Secure memory bandwidth usage monitor instances. MPAMF_MBWUMON_IDR_ns indicates the number of Non-secure memory bandwidth usage monitor instances.

Configurations

The power domain of MPAMF_MBWUMON_IDR is IMPLEMENTATION DEFINED.

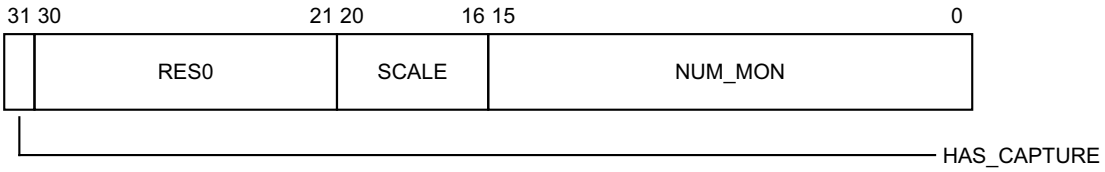
This register is present only when MPAMF_IDR.HAS_MSMON == 1 and MPAMF_MSMON_IDR.MSMON_MBWU == 1. Otherwise, direct accesses to MPAMF_MBWUMON_IDR are RES0.

Attributes

MPAMF_MBWUMON_IDR is a 32-bit register.

Field descriptions

The MPAMF_MBWUMON_IDR bit assignments are:



HAS_CAPTURE, bit [31]

The MSC's implementation supports copying an MSMON_MBWU to the corresponding MSMON_MBWU_CAPTURE on a capture event.

`0b0` MSMON_MBWU_CAPTURE is not implemented and there is no support for capture events in this component's MBWU monitor.

`0b1` The MSMON_MBWU_CAPTURE register is implemented and this component's MBWU monitor supports the capture event behavior.

Bits [30:21]

Reserved, RES0.

SCALE, bits [20:16]

Scaling of MSMON_MBWU.VALUE in bits. If scaling is enabled by MSMON_CFG_MBWU_CTL.SCLEN, the byte count in the VALUE field has been shifted by SCALE bits to the right.

`0b00000` Scaling is not implemented.

`0bxxxxx` Other values are right shift count when scaling is enabled.

NUM_MON, bits [15:0]

The number of memory bandwidth usage monitors implemented in this MSC.

Accessing the MPAMF_MBWUMON_IDR:

This register is within the MPAM feature page memory frames. In a system that supports Secure and Non-secure memory maps, there must be both Secure and Non-secure MPAM feature pages.

MPAMF_MBWUMON_IDR is read-only.

MPAMF_MBWUMON_IDR must be readable from the Non-secure and Secure MPAM feature pages.

MPAMF_MBWUMON_IDR is permitted to have the same contents when read from either the Secure and Non-secure MPAM feature pages unless the register contents is different for Secure and Non-secure versions, when there must be separate registers in the Secure (MPAMF_MBWUMON_IDR_s) and Non-secure (MPAMF_MBWUMON_IDR_ns) MPAM feature pages.

MPAMF_MBWUMON_IDR can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_s	0x0090	MPAMF_MBWUMON_IDR_s

This interface is accessible as follows:

- Accesses to this register are RO.

MPAMF_MBWUMON_IDR can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_ns	0x0090	MPAMF_MBWUMON_IDR_ns

This interface is accessible as follows:

- Accesses to this register are RO.

11.3.10 MPAMF_MSMON_IDR, MPAM Resource Monitoring Identification Register

The MPAMF_MSMON_IDR characteristics are:

Purpose

The MPAMF_MSMON_IDR is a 32-bit read-only register that indicates which MPAM monitoring features are present on this MSC. MPAMF_MSMON_IDR_s indicates Secure monitoring features. MPAMF_MSMON_IDR_ns indicates Non-secure monitoring features.

Configurations

The power domain of MPAMF_MSMON_IDR is IMPLEMENTATION DEFINED.

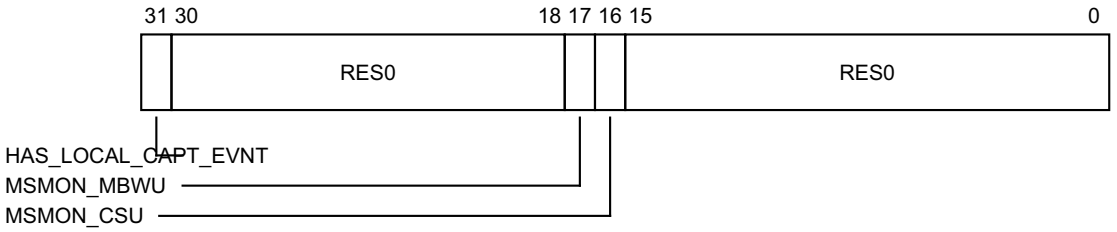
This register is present only when MPAMF_IDR.HAS_MSMON == 1. Otherwise, direct accesses to MPAMF_MSMON_IDR are RES0.

Attributes

MPAMF_MSMON_IDR is a 32-bit register.

Field descriptions

The MPAMF_MSMON_IDR bit assignments are:



HAS_LOCAL_CAPT_EVNT, bit [31]

Has local capture event generator. Indicates whether this MSC has the MPAM local capture event generator and the MSMON_CAPT_EVNT register.

`0b0` Does not support MPAM local capture event generator or MSMON_CAPT_EVNT.

`0b1` Supports the MPAM local capture event generator and the MSMON_CAPT_EVNT register.

Bits [30:18]

Reserved, RES0.

MSMON_MBWU, bit [17]

Memory bandwidth usage monitoring. Indicates whether this MSC has MPAM monitoring for Memory Bandwidth Usage by PARTID and PMG.

`0b0` Does not have monitoring for memory bandwidth usage or the MPAMF_MBWUMON_IDR, MSMON_CFG_MBWU_CTL, MSMON_CFG_MBWU_FLT, MSMON_MBWU or MSMON_MBWU_CAPTURE registers.

`0b1` Has monitoring of memory bandwdith usage and the MPAMF_MBWUMON_IDR, MSMON_CFG_MBWU_CTL, MSMON_CFG_MBWU_FLT, MSMON_MBWU and optional MSMON_MBWU_CAPTURE registers.

MSMON_CSU, bit [16]

Cache storage usage monitoring. Indicates whether this MSC has MPAM monitoring of cache storage usage by PARTID and PMG.

`0b0` Does not have monitoring for cache storage usage or the MPAMF_CSUMON_IDR, MSMON_CFG_CSU_CTL, MSMON_CFG_CSU_FLT, MSMON_CSU or MSMON_CSU_CAPTURE registers.

`0b1` Has monitoring of cache storage usage and the MPAMF_CSUMON_IDR, MSMON_CFG_CSU_CTL, MSMON_CFG_CSU_FLT, MSMON_CSU and optional MSMON_CSU_CAPTURE registers.

Bits [15:0]

Reserved, RES0.

Accessing the MPAMF_MSMON_IDR:

This register is within the MPAM feature page memory frames. In a system that supports Secure and Non-secure memory maps, there must be both Secure and Non-secure MPAM feature pages.

MPAMF_MSMON_IDR is read-only.

MPAMF_MSMON_IDR must be readable from the Non-secure and Secure MPAM feature pages.

MPAMF_MSMON_IDR is permitted to have the same contents when read from either the Secure and Non-secure MPAM feature pages unless the register contents is different for Secure and Non-secure versions, when there must be separate registers in the Secure (MPAMF_MSMON_IDR_s) and Non-secure (MPAMF_MSMON_IDR_ns) MPAM feature pages.

MPAMF_MSMON_IDR can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_s	`0x0080`	MPAMF_MSMON_IDR_s

This interface is accessible as follows:

- Accesses to this register are RO.

MPAMF_MSMON_IDR can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_ns	`0x0080`	MPAMF_MSMON_IDR_ns

This interface is accessible as follows:

- Accesses to this register are RO.

11.3.11 MPAMF_PARTID_NRW_IDR, MPAM PARTID Narrowing ID register

The MPAMF_PARTID_NRW_IDR characteristics are:

Purpose

The MPAMF_PARTID_NRW_IDR is a 32-bit read-only register that indicates the largest internal PARTID for this MSC. MPAMF_PARTID_NRW_IDR_s indicates the largest Secure internal PARTID. MPAMF_PARTID_NRW_IDR_ns indicates the largest Non-secure internal PARTID.

Configurations

The power domain of MPAMF_PARTID_NRW_IDR is IMPLEMENTATION DEFINED.

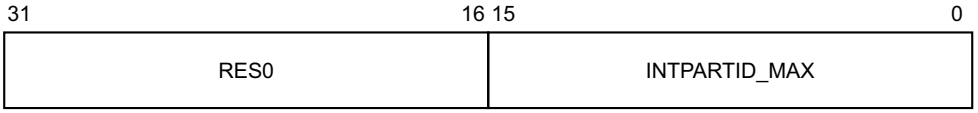
This register is present only when MPAMF_IDR.HAS_PARTID_NRW == 1. Otherwise, direct accesses to MPAMF_PARTID_NRW_IDR are RES0.

Attributes

MPAMF_PARTID_NRW_IDR is a 32-bit register.

Field descriptions

The MPAMF_PARTID_NRW_IDR bit assignments are:



Bits [31:16]

Reserved, RES0.

INTPARTID_MAX, bits [15:0]

The largest intPARTID supported in this MSC.

Accessing the MPAMF_PARTID_NRW_IDR:

This register is within the MPAM feature page memory frames. In a system that supports Secure and Non-secure memory maps, there must be both Secure and Non-secure MPAM feature pages.

MPAMF_PARTID_NRW_IDR is read-only.

MPAMF_PARTID_NRW_IDR must be readable from the Non-secure and Secure MPAM feature pages.

MPAMF_PARTID_NRW_IDR is permitted to have the same contents when read from either the Secure and Non-secure MPAM feature pages unless the register contents is different for Secure and Non-secure versions, when there must be separate registers in the Secure (MPAMF_PARTID_NRW_IDR_s) and Non-secure (MPAMF_PARTID_NRW_IDR_ns) MPAM feature pages.

MPAMF_PARTID_NRW_IDR can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_s	0x0050	MPAMF_PARTID_NRW_IDR_s

This interface is accessible as follows:

- Accesses to this register are RO.

MPAMF_PARTID_NRW_IDR can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_ns	0x0050	MPAMF_PARTID_NRW_IDR_ns

This interface is accessible as follows:

- Accesses to this register are RO.

11.3.12 MPAMF_PRI_IDR, MPAM Priority Partitioning Identification Register

The MPAMF_PRI_IDR characteristics are:

Purpose

The MPAMF_PRI_IDR is a 32-bit read-only register that indicates which MPAM priority partitioning features are present on this MSC. MPAMF_PRI_IDR_s indicates priority partitioning features accessed from the Secure MPAM feature page. MPAMF_PRI_IDR_ns indicates priority partitioning features accessed from the Non-secure MPAM feature page.

Configurations

The power domain of MPAMF_PRI_IDR is IMPLEMENTATION DEFINED.

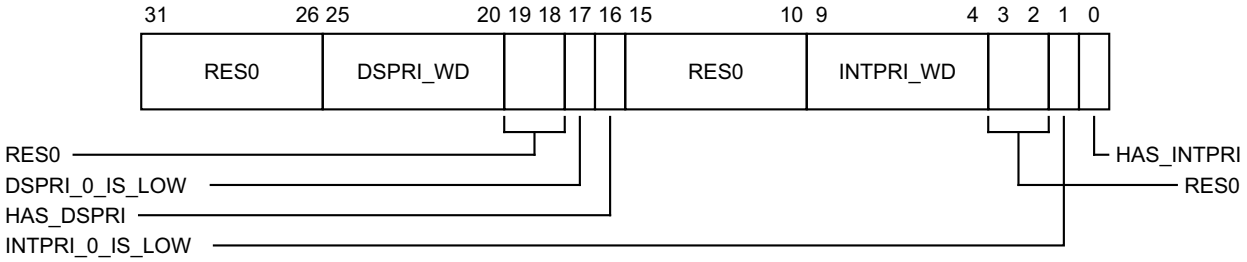
This register is present only when MPAMF_IDR.HAS_PRI_PART == 1. Otherwise, direct accesses to MPAMF_PRI_IDR are RES0.

Attributes

MPAMF_PRI_IDR is a 32-bit register.

Field descriptions

The MPAMF_PRI_IDR bit assignments are:



Bits [31:26]

Reserved, RES0.

DSPRI_WD, bits [25:20]

Number of implemented bits in the downstream priority field (DSPRI) of MPAMCFG_PRI.

If HAS_DSPRI == 1, this field must contain a value from 1 to 32, inclusive.

If HAS_DSPRI == 0, this field must be 0.

Bits [19:18]

Reserved, RES0.

DSPRI_0_IS_LOW, bit [17]

Indicates whether 0 in MPAMCFG_PRI.DSPRI is the lowest or the highest priority.

`0b0` In the MPAMCFG_PRI.DSPRI field, a value of 0 means the highest priority.

`0b1` In the MPAMCFG_PRI.DSPRI field, a value of 0 means the lowest priority.

HAS_DSPRI, bit [16]

Indicates that this MSC implements the DSPRI field in the MPAMCFG_PRI register.

`0b0` This MSC supports priority partitioning, but does not implement a downstream priority (DSPRI) field in the MPAMCFG_PRI register.

`0b1` This MSC supports downstream priority partitioning and implements the downstream priority (DSPRI) field in the MPAMCFG_PRI register.

Bits [15:10]

Reserved, RES0.

INTPRI_WD, bits [9:4]

Number of implemented bits in the internal priority field (INTPRI) in the MPAMCFG_PRI register.

If HAS_INTPRI == 1, this field must contain a value from 1 to 32, inclusive.

If HAS_INTPRI == 0, this field must be 0.

Bits [3:2]

Reserved, RES0.

INTPRI_0_IS_LOW, bit [1]

Indicates whether 0 in MPAMCFG_PRI.INTPRI is the lowest or the highest priority.

`0b0` In the MPAMCFG_PRI.INTPRI field, a value of 0 means the highest priority.

`0b1` In the MPAMCFG_PRI.INTPRI field, a value of 0 means the lowest priority.

HAS_INTPRI, bit [0]

Indicates that this MSC implements the INTPRI field in the MPAMCFG_PRI register.

`0b0` This MSC supports priority partitioning, but does not implement the internal priority (INTPRI) field in the MPAMCFG_PRI register.

`0b1` This MSC supports internal priority partitioning and implements the internal priority (INTPRI) field in the MPAMCFG_PRI register.

Accessing the MPAMF_PRI_IDR:

This register is within the MPAM feature page memory frames. In a system that supports Secure and Non-secure memory maps, there must be both Secure and Non-secure MPAM feature pages.

MPAMF_PRI_IDR is read-only.

MPAMF_PRI_IDR must be readable from the Non-secure and Secure MPAM feature pages.

MPAMF_PRI_IDR is permitted to have the same contents when read from either the Secure and Non-secure MPAM feature pages unless the register contents is different for Secure and Non-secure versions, when there must be separate registers in the Secure (MPAMF_PRI_IDR_s) and Non-secure (MPAMF_PRI_IDR_ns) MPAM feature pages.

MPAMF_PRI_IDR can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_s	`0x0048`	MPAMF_PRI_IDR_s

This interface is accessible as follows:

- Accesses to this register are RO.

MPAMF_PRI_IDR can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_ns	`0x0048`	MPAMF_PRI_IDR_ns

This interface is accessible as follows:

- Accesses to this register are RO.

11.3.13 MPAMF_SIDR, MPAM Features Secure Identification Register

The MPAMF_SIDR characteristics are:

Purpose

The MPAMF_SIDR is a 32-bit read-only register that indicates the maximum Secure PARTID and Secure PMG on this MSC.

Configurations

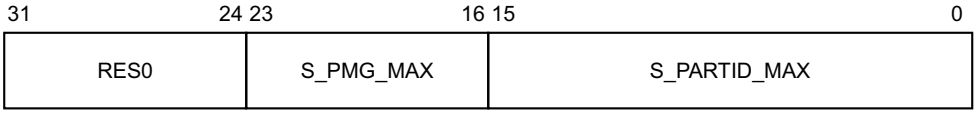
The power domain of MPAMF_SIDR is IMPLEMENTATION DEFINED.

Attributes

MPAMF_SIDR is a 32-bit register.

Field descriptions

The MPAMF_SIDR bit assignments are:



Bits [31:24]

Reserved, RES0.

S_PMG_MAX, bits [23:16]

Maximum value of Secure PMG supported by this component.

S_PARTID_MAX, bits [15:0]

Maximum value of Secure PARTID supported by this component.

Accessing the MPAMF_SIDR:

This register is within the MPAM feature page memory frames. In a system that supports Secure and Non-secure memory maps, there must be both Secure and Non-secure MPAM feature pages.

MPAMF_SIDR is read-only.

MPAMF_SIDR must only be readable from the Secure MPAM feature page. If the system or the MSC does not support the Secure address map, this register must not be accessible.

MPAMF_SIDR can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_s	0x0008	MPAMF_SIDR_s

This interface is accessible as follows:

- Accesses to this register are RO.

11.4 Memory-mapped partitioning configuration registers

This section lists the external partitioning configuration registers.

11.4.1 MPAMCFG_CMAX, MPAM Cache Maximum Capacity Partition Configuration Register

The MPAMCFG_CMAX characteristics are:

Purpose

The MPAMCFG_CMAX is a 32-bit read-write register that controls the maximum fraction of the cache capacity that the PARTID selected by MPAMCFG_PART_SEL is permitted to allocate. MPAMCFG_CMAX_s controls cache maximum capacity for the Secure PARTID selected by the Secure instance of MPAMCFG_PART_SEL. MPAMCFG_CMAX_ns controls the cache maximum capacity for the Non-secure PARTID selected by the Non-secure instance of MPAMCFG_PART_SEL.

Configurations

The power domain of MPAMCFG_CMAX is IMPLEMENTATION DEFINED.

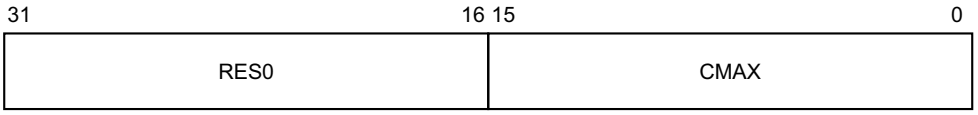
This register is present only when MPAMF_IDR.HAS_CCAP_PART == 1. Otherwise, direct accesses to MPAMCFG_CMAX are RES0.

Attributes

MPAMCFG_CMAX is a 32-bit register.

Field descriptions

The MPAMCFG_CMAX bit assignments are:



Bits [31:16]

Reserved, RES0.

CMAX, bits [15:0]

Maximum cache capacity usage in fixed-point fraction format by the partition selected by MPAMCFG_PART_SEL. The fraction represents the portion of the total cache capacity that the PARTID is permitted to allocate.

The implemented width of the fixed-point fraction is given in MPAMF_CCAP_IDR.CMAX_WD. Unimplemented bits within the field are RAZ/WI. The implemented bits of the CMAX field are always the most-significant bits of the field.

The fixed-point fraction CMAX is less than 1. The implied binary point is between bits 15 and 16. This representation has as the largest fraction of the cache that can be represented in an implementation with w implemented bits is 1 - 1/w.

Accessing the MPAMCFG_CMAX:

This register is within the MPAM feature page memory frames. In a system that supports Secure and Non-secure memory maps, there must be both Secure and Non-secure MPAM feature pages.

MPAMCFG_CMAX_s must be accessible from the Secure MPAM feature page. MPAMCFG_CMAX_ns must be accessible from the Non-secure MPAM feature page.

MPAMCFG_CMAX_s and MPAMCFG_CMAX_ns must be separate registers. The Secure instance (MPAMCFG_CMAX_s) accesses the cache capacity partitioning used for Secure PARTIDs, and the Non-secure instance (MPAMCFG_CMAX_ns) accesses the cache capacity partitioning used for Non-secure PARTIDs.

MPAMCFG_CMAX can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_ns	0x0108	MPAMCFG_CMAX_ns

This interface is accessible as follows:

- Accesses to this register are RW.

MPAMCFG_CMAX can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_s	0x0108	MPAMCFG_CMAX_s

This interface is accessible as follows:

- Accesses to this register are RW.

11.4.2 MPAMCFG_CPBM, MPAM Cache Portion Bitmap Partition Configuration Register

The MPAMCFG_CPBM characteristics are:

Purpose

The MPAMCFG_CPBM register is a read-write register that configures the cache portions that a PARTID is allowed to allocate. After setting MPAMCFG_PART_SEL with a PARTID, software (usually a hypervisor) writes to the MPAMCFG_CPBM register to configure which cache portions the PARTID is allowed to allocate.

MPAMCFG_CPBM_s controls cache portions for the Secure PARTID selected by the Secure instance of MPAMCFG_PART_SEL. MPAMCFG_CPBM_ns controls the cache portions for the Non-secure PARTID selected by the Non-secure instance of MPAMCFG_PART_SEL.

Configurations

The power domain of MPAMCFG_CPBM is IMPLEMENTATION DEFINED.

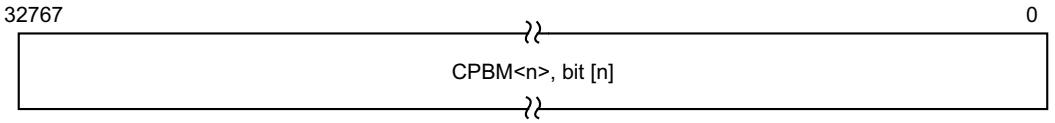
This register is present only when MPAMF_IDR.HAS_CPOR == 1. Otherwise, direct accesses to MPAMCFG_CPBM are RES0.

Attributes

MPAMCFG_CPBM is a 32768-bit register.

Field descriptions

The MPAMCFG_CPBM bit assignments are:



CPBM<n>, bit [n], for n = 0 to 32767

Each bit, CPBM<n>, grants permission to the PARTID to allocate cache lines within cache portion n.

`0b0` The PARTID is not permitted to allocate into cache portion n.

`0b1` The PARTID is permitted to allocate within cache portion n.

The number of bits in the cache portion partitioning bit map of this component is given in MPAMF_CPOR_IDR.CPBM_WD. CPBM_WD contains a value from 1 to 2^{15}, inclusive. Values of CPBM_WD greater than 32 require a group of 32-bit registers to access the CPBM, up to 1024 registers.

Bits CPBM<n>, where n is greater than CPBM_WD, are not required to be implemented.

Accessing the MPAMCFG_CPBM:

This register is within the MPAM feature page memory frames. In a system that supports Secure and Non-secure memory maps, there must be both Secure and Non-secure MPAM feature pages.

MPAMCFG_CPBM_s must be accessible from the Secure MPAM feature page. MPAMCFG_CPBM_ns must be accessible from the Non-secure MPAM feature page.

MPAMCFG_CPBM_s and MPAMCFG_CPBM_ns must be separate registers. The Secure instance (MPAMCFG_CPBM_s) accesses the cache portion bitmaps used for Secure PARTIDs, and the Non-secure instance (MPAMCFG_CPBM_ns) accesses the cache portion bitmaps used for Non-secure PARTIDs.

MPAMCFG_CPBM can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_s	0x1000	MPAMCFG_CPBM_s

This interface is accessible as follows:

- Accesses to this register are RW.

MPAMCFG_CPBM can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_ns	0x1000	MPAMCFG_CPBM_ns

This interface is accessible as follows:

- Accesses to this register are RW.

11.4.3 MPAMCFG_INTPARTID, MPAM Internal PARTID Narrowing Configuration Register

The MPAMCFG_INTPARTID characteristics are:

Purpose

MPAMCFG_INTPARTID is a 32-bit read-write register that controls the mapping of the PARTID selected by MPAMCFG_PART_SEL into a narrower internal PARTID (intPARTID).

MPAMCFG_INTPARTID_s controls the mapping for the Secure PARTID selected by the Secure instance of MPAMCFG_PART_SEL. MPAMCFG_INTPARTID_ns controls the mapping for the Non-secure PARTID selected by the Non-secure instance of MPAMCFG_PART_SEL.

The MPAMCFG_INTPARTID register associates the request PARTID (reqPARTID) in the MPAMCFG_PART_SEL register with an internal PARTID (intPARTID) in this register. To set that association, store reqPARTID into the MPAMCFG_PART_SEL register and then store the intPARTID into the MPAMCFG_INTPARTID register. To read the association, store reqPARTID into the MPAMCFG_PART_SEL register and then read MPAMCFG_INTPARTID.

If the intPARTID stored into MPAMCFG_INTPARTID is out-of-range or does not have the INTERNAL bit set, the association of reqPARTID to intPARTID is not written and MPAMF_ESR is set to indicate an intPARTID_Range error.

If MPAMCFG_PART_SEL.INTERNAL is 1 when MPAMCFG_INTPARTID is read or written, MPAMF_ESR is set to indicate an unexpected_internal error.

Configurations

The power domain of MPAMCFG_INTPARTID is IMPLEMENTATION DEFINED.

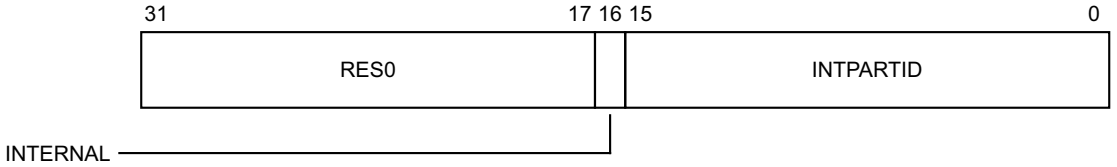
This register is present only when MPAMF_IDR.HAS_PARTID_NRW == 1. Otherwise, direct accesses to MPAMCFG_INTPARTID are RES0.

Attributes

MPAMCFG_INTPARTID is a 32-bit register.

Field descriptions

The MPAMCFG_INTPARTID bit assignments are:



Bits [31:17]

Reserved, RES0.

INTERNAL, bit [16]

Internal PARTID flag.

This bit must be 1 when written to the register. If written as 0, the write will not update the reqPARTID to intPARTID association.

On a read of this register, the bit will always read the value last written.

INTPARTID, bits [15:0]

This field contains the intPARTID mapped to the reqPARTID in MPAMCFG_PART_SEL.

The maximum intPARTID supported is MPAMF_PARTID_NRW_IDR.INTPARTID_MAX.

Accessing the MPAMCFG_INTPARTID:

This register is within the MPAM feature page memory frames. In a system that supports Secure and Non-secure memory maps, there must be both Secure and Non-secure MPAM feature pages.

MPAMCFG_INTPARTID_s must be accessible from the Secure MPAM feature page.
MPAMCFG_INTPARTID_ns must be accessible from the Non-secure MPAM feature page.

MPAMCFG_INTPARTID_s and MPAMCFG_INTPARTID_ns must be separate registers. The Secure instance (MPAMCFG_INTPARTID_s) accesses the PARTID narrowing used for Secure PARTIDs, and the Non-secure instance (MPAMCFG_INTPARTID_ns) accesses the PARTID narrowing used for Non-secure PARTIDs.

MPAMCFG_INTPARTID can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_s	0x0600	MPAMCFG_INTPARTID_s

This interface is accessible as follows:

- Accesses to this register are RW.

MPAMCFG_INTPARTID can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_ns	0x0600	MPAMCFG_INTPARTID_ns

This interface is accessible as follows:

- Accesses to this register are RW.

11.4.4 MPAMCFG_MBW_MAX, MPAM Memory Bandwidth Maximum Partition Configuration Register

The MPAMCFG_MBW_MAX characteristics are:

Purpose

MPAMCFG_MBW_MAX is a 32-bit read-write register that controls the maximum fraction of memory bandwidth that the PARTID selected by MPAMCFG_PART_SEL is permitted to use. MPAMCFG_MBW_MAX_s controls maximum bandwidth for the Secure PARTID selected by the Secure instance of MPAMCFG_PART_SEL. MPAMCFG_MBW_MAX_ns controls the maximum bandwidth for the Non-secure PARTID selected by the Non-secure instance of MPAMCFG_PART_SEL.

A PARTID that has used more than MAX is given no access to additional bandwidth if HARDLIM == 1 or is given additional bandwidth only if there are no requests from PARTIDs that have not exceeded their MAX if HARDLIM == 0.

Configurations

The power domain of MPAMCFG_MBW_MAX is IMPLEMENTATION DEFINED.

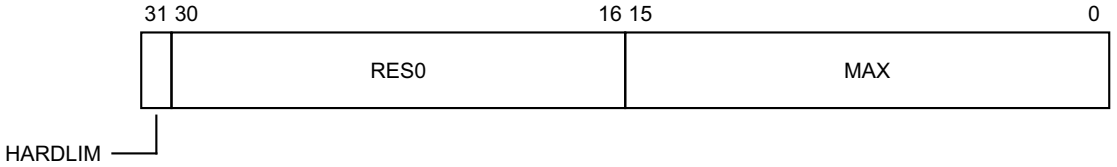
This register is present only when MPAMF_IDR.HAS_MBW_PART == 1 and MPAMF_MBW_IDR.HAS_MAX == 1. Otherwise, direct accesses to MPAMCFG_MBW_MAX are RES0.

Attributes

MPAMCFG_MBW_MAX is a 32-bit register.

Field descriptions

The MPAMCFG_MBW_MAX bit assignments are:



HARDLIM, bit [31]

Hard bandwidth limiting.

`0b0` When MAX bandwidth is exceeded, the partition contends with a low preference for downstream bandwidth beyond its maximum bandwidth.

`0b1` When MAX bandwidth is exceeded, the partition does not be use any more bandwidth until its memory bandwidth measurement falls below the maximum limit.

Bits [30:16]

Reserved, RES0.

MAX, bits [15:0]

Memory maximum bandwidth allocated to the partition selected by MPAMCFG_PART_SEL. MAX is in fixed-point fraction format. The fraction represents the portion of the total memory bandwidth capacity through the controlled component that the PARTID is permitted to allocate.

The implemented width of the fixed-point fraction is given in MPAMF_MBW_IDR.BWA_WD. Unimplemented bits are RAZ/WI. The implemented bits of the MAX field are always to the left of the field. For example, if BWA_WD = 3, the implemented bits are MPAMCFG_MBW_MAX[15:13] and MPAMCFG_MBW_MAX[12:0] are unimplemented.

The fixed-point fraction MAX is less than 1. The implied binary point is between bits 15 and 16. This representation has as the largest fraction of the cache that can be represented in an implementation with w implemented bits is 1 - 1/w.

Accessing the MPAMCFG_MBW_MAX:

This register is within the MPAM feature page memory frames. In a system that supports Secure and Non-secure memory maps, there must be both Secure and Non-secure MPAM feature pages.

MPAMCFG_MBW_MAX_s must be accessible from the Secure MPAM feature page.
MPAMCFG_MBW_MAX_ns must be accessible from the Non-secure MPAM feature page.

MPAMCFG_MBW_MAX_s and MPAMCFG_MBW_MAX_ns must be separate registers. The Secure instance (MPAMCFG_MBW_MAX_s) accesses the memory maximum bandwidth partitioning used for Secure PARTIDs, and the Non-secure instance (MPAMCFG_MBW_MAX_ns) accesses the memory maximum bandwidth partitioning used for Non-secure PARTIDs.

MPAMCFG_MBW_MAX can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_s	0x0208	MPAMCFG_MBW_MAX_s

This interface is accessible as follows:

- Accesses to this register are RW.

MPAMCFG_MBW_MAX can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_ns	0x0208	MPAMCFG_MBW_MAX_ns

This interface is accessible as follows:

- Accesses to this register are RW.

11.4.5 MPAMCFG_MBW_MIN, MPAM Cache Maximum Capacity Partition Configuration Register

The MPAMCFG_MBW_MIN characteristics are:

Purpose

MPAMCFG_MBW_MIN is a 32-bit read-write register that controls the minimum fraction of memory bandwidth that the PARTID selected by MPAMCFG_PART_SEL is permitted to use. MPAMCFG_MBW_MIN_s controls the minimum bandwidth for the Secure PARTID selected by the Secure instance of MPAMCFG_PART_SEL. MPAMCFG_MBW_MIN_ns controls the minimum bandwidth for the Non-secure PARTID selected by the Non-secure instance of MPAMCFG_PART_SEL.

A PARTID that has used less than MIN is given preferential access to bandwidth.

Configurations

The power domain of MPAMCFG_MBW_MIN is IMPLEMENTATION DEFINED.

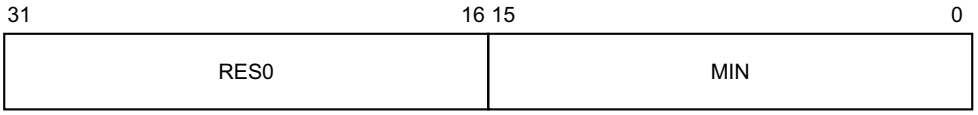
This register is present only when MPAMF_IDR.HAS_MBW_PART == 1 and MPAMF_MBW_IDR.HAS_MIN == 1. Otherwise, direct accesses to MPAMCFG_MBW_MIN are RES0.

Attributes

MPAMCFG_MBW_MIN is a 32-bit register.

Field descriptions

The MPAMCFG_MBW_MIN bit assignments are:



Bits [31:16]

Reserved, RES0.

MIN, bits [15:0]

Memory minimum bandwidth allocated to the partition selected by MPAMCFG_PART_SEL. MIN is in fixed-point fraction format. The fraction represents the portion of the total memory bandwidth capacity through the controlled component that the PARTID is permitted to allocate.

The implemented width of the fixed-point fraction is given in MPAMF_MBW_IDR.BWA_WD. Unimplemented bits are RAZ/WI. The implemented bits of the MIN field are always to the left of the field. For example, if BWA_WD = 4, the implemented bits are MPAMCFG_MBW_MIN[15:12] and MPAMCFG_MBW_MIN[11:0] are unimplemented.

The fixed-point fraction MIN is less than 1. The implied binary point is between bits 15 and 16. This representation has as the largest fraction of the cache that can be represented in an implementation with w implemented bits is 1 - 1/w.

Accessing the MPAMCFG_MBW_MIN:

This register is within the MPAM feature page memory frames. In a system that supports Secure and Non-secure memory maps, there must be both Secure and Non-secure MPAM feature pages.

MPAMCFG_MBW_MIN_s must be accessible from the Secure MPAM feature page. MPAMCFG_MBW_MIN_ns must be accessible from the Non-secure MPAM feature page.

MPAMCFG_MBW_MIN_s and MPAMCFG_MBW_MIN_ns must be separate registers. The Secure instance (MPAMCFG_MBW_MIN_s) accesses the memory minimum bandwidth partitioning used for Secure PARTIDs, and the Non-secure instance (MPAMCFG_MBW_MIN_ns) accesses the memory minimum bandwidth partitioning used for Non-secure PARTIDs.

MPAMCFG_MBW_MIN can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_s	0x0200	MPAMCFG_MBW_MIN_s

This interface is accessible as follows:

- Accesses to this register are RW.

MPAMCFG_MBW_MIN can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_ns	0x0200	MPAMCFG_MBW_MIN_ns

This interface is accessible as follows:

- Accesses to this register are RW.

11.4.6 MPAMCFG_MBW_PBM, MPAM Bandwidth Portion Bitmap Partition Configuration Register

The MPAMCFG_MBW_PBM characteristics are:

Purpose

The MPAMCFG_MBW_PBM register is a read-write register that configures the cache portions that a PARTID is allowed to allocate. MPAMCFG_MBW_PBM_s controls the bandwidth portion bitmap for the Secure PARTID selected by the Secure instance of MPAMCFG_PART_SEL. MPAMCFG_MBW_PBM_ns controls the bandwidth portion bitmap for the Non-secure PARTID selected by the Non-secure instance of MPAMCFG_PART_SEL.

After setting MPAMCFG_PART_SEL with a PARTID, software (usually a hypervisor) writes to the MPAMCFG_CPBM register to configure which cache portions the PARTID is allowed to allocate.

Configurations

The power domain of MPAMCFG_MBW_PBM is IMPLEMENTATION DEFINED.

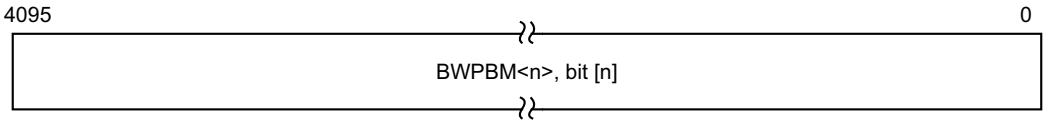
This register is present only when MPAMF_IDR.HAS_MBW_PART == 1 and MPAMF_MBW_IDR.HAS_PBM == 1. Otherwise, direct accesses to MPAMCFG_MBW_PBM are RES0.

Attributes

MPAMCFG_MBW_PBM is a 4096-bit register.

Field descriptions

The MPAMCFG_MBW_PBM bit assignments are:



BWPBM<n>, bit [n], for n = 0 to 4095

Each bit BWPBM<n> grants permission to the PARTID to allocate bandwidth within bandwidth portion n.

`0b0` The PARTID is not permitted to allocate into bandwidth portion n.

`0b1` The PARTID is permitted to allocate within bandwidth portion n.

The number of bits in the bandwidth portion partitioning bit map of this component is given in MPAMF_MBW_IDR.BWPBM_WD. BWPBM_WD contains a value from 1 to 2^{12}, inclusive. Values of BWPBM_WD greater than 32 require a group of 32-bit registers to access the BWPBM, up to 128 32-bit registers.

Bits BWPBM<n>, where n is greater than BWPBM_WD, are not required to be implemented.

Accessing the MPAMCFG_MBW_PBM:

This register is within the MPAM feature page memory frames. In a system that supports Secure and Non-secure memory maps, there must be both Secure and Non-secure MPAM feature pages.

MPAMCFG_MBW_PBM_s must be accessible from the Secure MPAM feature page.
MPAMCFG_MBW_PBM_ns must be accessible from the Non-secure MPAM feature page.

MPAMCFG_MBW_PBM_s and MPAMCFG_MBW_PBM_ns must be separate registers. The Secure instance (MPAMCFG_MBW_PBM_s) accesses the memory bandwidth portion bitmaps used for Secure PARTIDs, and the Non-secure instance (MPAMCFG_MBW_PBM_ns) accesses the memory bandwidth portion bitmaps used for Non-secure PARTIDs.

MPAMCFG_MBW_PBM can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_s	0x2000	MPAMCFG_MBW_PBM_s

This interface is accessible as follows:

- Accesses to this register are RW.

MPAMCFG_MBW_PBM can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_ns	0x2000	MPAMCFG_MBW_PBM_ns

This interface is accessible as follows:

- Accesses to this register are RW.

11.4.7 MPAMCFG_MBW_PROP, MPAM Memory Bandwidth Proportional Stride Partition Configuration Register

The MPAMCFG_MBW_PROP characteristics are:

Purpose

MPAMCFG_MBW_PROP is a 32-bit read-write register that controls the maximum fraction of memory bandwidth that the PARTID selected by MPAMCFG_PART_SEL is permitted to use. MPAMCFG_MBW_PROP_s controls the bandwidth proportional stride for the Secure PARTID selected by the Secure instance of MPAMCFG_PART_SEL. MPAMCFG_MBW_PROP_ns controls the bandwidth proportional stride for the Non-secure PARTID selected by the Non-secure instance of MPAMCFG_PART_SEL.

A PARTID that has a current bandwidth usage of more than MAX is prevented from access to additional bandwidth if HARDLIM == 1 or if HARDLIM == 0, the PARTID is given additional bandwidth only if there are no requests from PARTIDs that have not exceeded their MAX.

Configurations

The power domain of MPAMCFG_MBW_PROP is IMPLEMENTATION DEFINED.

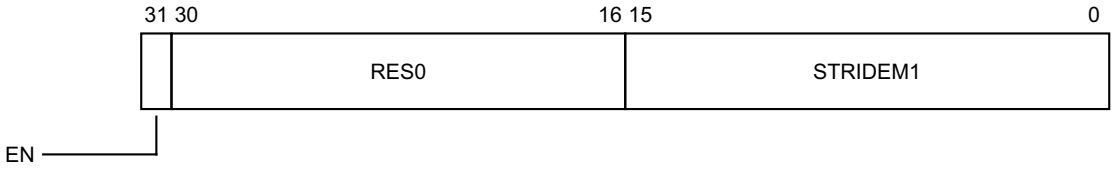
This register is present only when MPAMF_IDR.HAS_MBW_PART == 1 and MPAMF_MBW_IDR.HAS_PROP == 1. Otherwise, direct accesses to MPAMCFG_MBW_PROP are RES0.

Attributes

MPAMCFG_MBW_PROP is a 32-bit register.

Field descriptions

The MPAMCFG_MBW_PROP bit assignments are:



EN, bit [31]

Enable proportional stride bandwidth partitioning.

`0b0` The selected partition is not regulated by proportional stride bandwidth partitioning.

`0b1` The selected partition has bandwidth usage regulated by proportional stride bandwidth partitioning as controlled by STRIDEM1.

Bits [30:16]

Reserved, RES0.

STRIDEM1, bits [15:0]

Memory bandwidth stride minus 1 allocated to the partition selected by MPAMCFG_PART_SEL. STRIDEM1 represents the normalized cost of bandwidth consumption by the partition.

The proportional stride partitioning control parameter is an unsigned integer representing the normalized cost to a partition for consuming bandwidth. Larger values have a larger cost and correspond to a lesser allocation of bandwidth while smaller values indicate a lesser cost and therefore a higher allocation of bandwidth.

The implemented width of STRIDEM1 is given in MPAMF_MBW_IDR.BWA_WD.

Accessing the MPAMCFG_MBW_PROP:

This register is within the MPAM feature page memory frames. In a system that supports Secure and Non-secure memory maps, there must be both Secure and Non-secure MPAM feature pages.

MPAMCFG_MBW_PROP_s must be accessible from the Secure MPAM feature page.
MPAMCFG_MBW_PROP_ns must be accessible from the Non-secure MPAM feature page.

MPAMCFG_MBW_PROP_s and MPAMCFG_MBW_PROP_ns must be separate registers. The Secure instance (MPAMCFG_MBW_PROP_s) accesses the memory proportional stride bandwidth partitioning used for Secure PARTIDs, and the Non-secure instance (MPAMCFG_MBW_PROP_ns) accesses the memory proportional stride bandwidth partitioning used for Non-secure PARTIDs.

MPAMCFG_MBW_PROP can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_s	0x0500	MPAMCFG_MBW_PROP_s

This interface is accessible as follows:

- Accesses to this register are RW.

MPAMCFG_MBW_PROP can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_ns	0x0500	MPAMCFG_MBW_PROP_ns

This interface is accessible as follows:

- Accesses to this register are RW.

11.4.8 MPAMCFG_MBW_WINWD, MPAM Memory Bandwidth Partitioning Window Width Configuration Register

The MPAMCFG_MBW_WINWD characteristics are:

Purpose

MPAMCFG_MBW_WINWD is a 32-bit register that shows and sets the value of the window width for the PARTID in MPAMCFG_PART_SEL. MPAMCFG_MBW_WINWD_s reads and controls the bandwidth control window width for the Secure PARTID selected by the Secure instance of MPAMCFG_PART_SEL. MPAMCFG_MBW_WINWD_ns reads and controls the bandwidth control window for the Non-secure PARTID selected by the Non-secure instance of MPAMCFG_PART_SEL.

MPAMCFG_MBW_WINWD is read-only if MPAMF_MBW_IDR.WINDWR == 0, and the window width is set by the hardware, even if variable.

MPAMCFG_MBW_WINWD is read-write if MPAMF_MBW_IDR.WINDWR == 1, permitting configuration of the window width for each PARTID independently on hardware that supports this functionality.

Configurations

The power domain of MPAMCFG_MBW_WINWD is IMPLEMENTATION DEFINED.

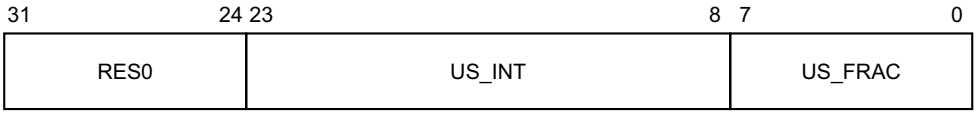
This register is present only when MPAMF_IDR.HAS_MBW_PART == 1. Otherwise, direct accesses to MPAMCFG_MBW_WINWD are RES0.

Attributes

MPAMCFG_MBW_WINWD is a 32-bit register.

Field descriptions

The MPAMCFG_MBW_WINWD bit assignments are:



Bits [31:24]

Reserved, RES0.

US_INT, bits [23:8]

Window width, integer microseconds.

This field reads (and sets) the integer part of the window width in microseconds for the PARTID selected by MPAMCFG_PART_SEL.

US_FRAC, bits [7:0]

Window width, fractional microseconds.

This field reads (and sets) the fractional part of the window width in microseconds for the PARTID selected by MPAMCFG_PART_SEL.

Accessing the MPAMCFG_MBW_WINWD:

This register is within the MPAM feature page memory frames. In a system that supports Secure and Non-secure memory maps, there must be both Secure and Non-secure MPAM feature pages.

MPAMCFG_MBW_WINWD_s must be accessible from the Secure MPAM feature page.
MPAMCFG_MBW_WINWD_ns must be accessible from the Non-secure MPAM feature page.

MPAMCFG_MBW_WINWD_s and MPAMCFG_MBW_WINWD_ns must be separate registers. The Secure instance (MPAMCFG_MBW_WINWD_s) accesses the window width used for Secure PARTIDs, and the Non-secure instance (MPAMCFG_MBW_WINWD_ns) accesses the window width used for Non-secure PARTIDs.

MPAMCFG_MBW_WINWD can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_s	0x0220	MPAMCFG_MBW_WINWD_s

This interface is accessible as follows:

- When MPAMF_MBW_IDR.WINWR == 0 accesses to this register are RO.
- Otherwise accesses to this register are RW.

MPAMCFG_MBW_WINWD can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_ns	0x0220	MPAMCFG_MBW_WINWD_ns

This interface is accessible as follows:

- When MPAMF_MBW_IDR.WINWR == 0 accesses to this register are RO.
- Otherwise accesses to this register are RW.

11.4.9 MPAMCFG_PART_SEL, MPAM Partition Configuration Selection Register

The MPAMCFG_PART_SEL characteristics are:

Purpose

The MPAMCFG_PART_SEL register is a 32-bit read-write register that selects a partition ID to configure. MPAMCFG_PART_SEL_s selects a Secure PARTID to configure. MPAMCFG_PART_SEL_ns selects a Non-secure PARTID to configure.

After setting this register with a PARTID, software (usually a hypervisor) can perform a series of accesses to MPAMCFG registers to configure parameters for MPAM resource controls to use when requests have that PARTID.

Configurations

The power domain of MPAMCFG_PART_SEL is IMPLEMENTATION DEFINED.

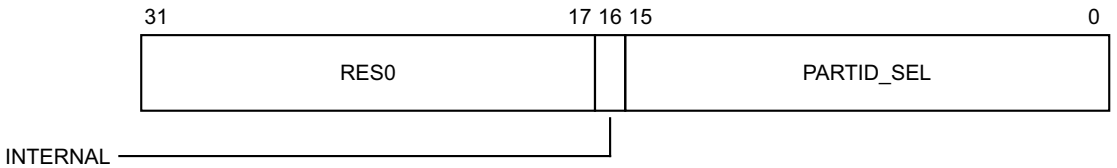
This register is present only when MPAMF_IDR.HAS_CCAP_PART == 1, or MPAMF_IDR.HAS_CPOR_PART == 1, or MPAMF_IDR.HAS_MBW_PART == 1, or MPAMF_IDR.HAS_PRI_PART == 1, or MPAMF_IDR.HAS_PARTID_NRW == 1 or MPAMF_IDR.HAS_IMPL_IDR == 1. Otherwise, direct accesses to MPAMCFG_PART_SEL are RES0.

Attributes

MPAMCFG_PART_SEL is a 32-bit register.

Field descriptions

The MPAMCFG_PART_SEL bit assignments are:



Bits [31:17]

Reserved, RES0.

INTERNAL, bit [16]

Internal PARTID.

If MPAMF_IDR.HAS_PARTID_NRW =0, this field is RAZ/WI.

If MPAMF_IDR.HAS_PARTID_NRW = 1:

`0b0` PARTID_SEL is interpreted as a request PARTID and ignored except for use with MPAMCFG_INTPARTID register access.

`0b1` PARTID_SEL is interpreted as an internal PARTID and used for access to MPAMCFG control settings except for MPAMCFG_INTPARTID.

If PARTID narrowing is implemented as indicated by MPAMF_IDR.HAS_PARTID_NRW = 1, when accessing other MPAMCFG registers when the value of the MPAMCFG_PART_SEL.INTERNAL bit is checked for these conditions:

- When the MPAMCFG_INTPARTID register is read or written, if the value of MPAMCFG_PART_SEL.INTERNAL is not 0, an Unexpected_INTERNAL error is set in MPAMF_ESR.
- When an MPAMCFG register other than MPAMCFG_INTPARTID is read or written, if the value of MPAMCFG_PART_SEL.INTERNAL is not 1, MPAMF_ESR is set to indicate an intPARTID_Range error.

In either error case listed here, the value returned by a read operation is UNPREDICTABLE, and the control settings are not affected by a write.

PARTID_SEL, bits [15:0]

Selects the partition ID to configure.

Reads and writes to other MPAMCFG registers are indexed by PARTID_SEL and by the NS bit used to access MPAMCFG_PART_SEL to access the configuration for a single partition.

Accessing the MPAMCFG_PART_SEL:

This register is within the MPAM feature page memory frames. In a system that supports Secure and Non-secure memory maps, there must be both Secure and Non-secure MPAM feature pages.

MPAMCFG_PART_SEL_s must be accessible from the Secure MPAM feature page. MPAMCFG_PART_SEL_ns must be accessible from the Non-secure MPAM feature page.

MPAMCFG_PART_SEL_s and MPAMCFG_PART_SEL_ns must be separate registers. The Secure instance (MPAMCFG_PART_SEL_s) accesses the PARTID selector used for Secure PARTIDs, and the Non-secure instance (MPAMCFG_PART_SEL_ns) accesses the PARTID selector used for Non-secure PARTIDs.

MPAMCFG_PART_SEL can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_s	0x0100	MPAMCFG_PART_SEL_s

This interface is accessible as follows:

- Accesses to this register are RW.

MPAMCFG_PART_SEL can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_ns	0x0100	MPAMCFG_PART_SEL_ns

This interface is accessible as follows:

- Accesses to this register are RW.

11.4.10 MPAMCFG_PRI, MPAM Priority Partition Configuration Register

The MPAMCFG_PRI characteristics are:

Purpose

MPAMCFG_PRI is a 32-bit read-write register that controls the internal and downstream priority of requests attributed to the PARTID selected by MPAMCFG_PART_SEL. MPAMCFG_PRI_s controls the priorities for the Secure PARTID selected by the Secure instance of MPAMCFG_PART_SEL. MPAMCFG_PRI_ns controls the priorities for the Non-secure PARTID selected by the Non-secure instance of MPAMCFG_PART_SEL.

Configurations

The power domain of MPAMCFG_PRI is IMPLEMENTATION DEFINED.

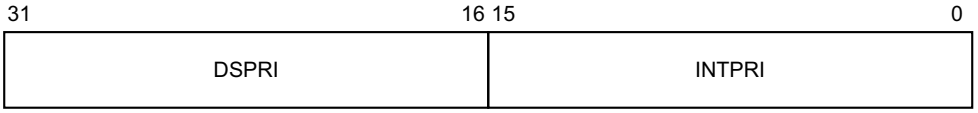
This register is present only when MPAMF_IDR.HAS_PRI_PART == 1. Otherwise, direct accesses to MPAMCFG_PRI are RES0.

Attributes

MPAMCFG_PRI is a 32-bit register.

Field descriptions

The MPAMCFG_PRI bit assignments are:



DSPRI, bits [31:16]

Downstream priority.

If MPAMF_PRI_IDR.HAS_DSPRI == 0, bits of this field are RES0, as this field is not used.

If MPAMF_PRI_IDR.HAS_DSPRI == 1, this field is a priority value applied to downstream communications from this MSC for transactions of the partition selected by MPAMCFG_PART_SEL.

The implemented width of this field is MPAMF_PRI_IDR.DSPRI_WD bits.

The encoding of priority is 0-as-lowest or 0-as-highest priority according to the value of MPAMF_PRI_IDR.DSPRI_0_IS_LOW.

INTPRI, bits [15:0]

Internal priority.

If MPAMF_PRI_IDR.HAS_INTPRI == 0, bits of this field are RES0 as this field is not used.

If MPAMF_PRI_IDR.HAS_INTPRI == 1, this field is a priority value applied internally inside this MSC for transactions of the partition selected by Mext-PAMCFG_PART_SEL.

The implemented width of this field is MPAMF_PRI_IDR.INTPRI_WD bits.

The encoding of priority is 0-as-lowest or 0-as-highest priority according to the value of MPAMF_PRI_IDR.INTPRI_0_IS_LOW.

Accessing the MPAMCFG_PRI:

This register is within the MPAM feature page memory frames. In a system that supports Secure and Non-secure memory maps, there must be both Secure and Non-secure MPAM feature pages.

MPAMCFG_PRI_s must be accessible from the Secure MPAM feature page. MPAMCFG_PRI_ns must be accessible from the Non-secure MPAM feature page.

MPAMCFG_PRI_s and MPAMCFG_PRI_ns must be separate registers. The Secure instance (MPAMCFG_PRI_s) accesses the priority partitioning used for Secure PARTIDs, and the Non-secure instance (MPAMCFG_PRI_ns) accesses the priority partitioning used for Non-secure PARTIDs.

MPAMCFG_PRI can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_s	0x0400	MPAMCFG_PRI_s

This interface is accessible as follows:

- Accesses to this register are RW.

MPAMCFG_PRI can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_ns	0x0400	MPAMCFG_PRI_ns

This interface is accessible as follows:

- Accesses to this register are RW.

11.5 Memory-mapped monitoring configuration registers

This section lists the external monitoring configuration registers.

11.5.1 MSMON_CAPT_EVNT, MPAM Capture Event Generation Register

The MSMON_CAPT_EVNT characteristics are:

Purpose

MSMON_CAPT_EVNT is a 32-bit read-write register that generates a local capture event when written with bit 0 as 1. MSMON_CAPT_EVNT_s generates local capture events for Secure monitors only or for Secure and Non-secure monitors. MSMON_CAPT_EVNT_ns generates local capture events for Non-secure monitors only.

Configurations

The power domain of MSMON_CAPT_EVNT is IMPLEMENTATION DEFINED.

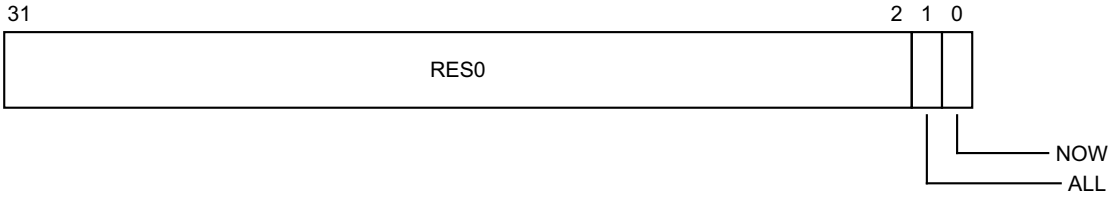
This register is present only when MPAMF_IDR.HAS_MSMON == 1 and MPAMF_MSMON_IDR.HAS_LOCAL_CAPT_EVNT == 1. Otherwise, direct accesses to MSMON_CAPT_EVNT are RES0.

Attributes

MSMON_CAPT_EVNT is a 32-bit register.

Field descriptions

The MSMON_CAPT_EVNT bit assignments are:



Bits [31:2]

Reserved, RES0.

ALL, bit [1]

In the Secure instance of this register, if ALL written as 1 and NOW is also written as 1, signal a capture event to Secure and Non-secure monitor instances in this MSC that are configured with CAPT_EVNT = 7.

If written as 0 and NOW is written as 1, signal a capture event to Secure monitor instances in this MSC that are configured with CAPT_EVNT = 7.

In the Non-secure instance of this register, this bit is RAZ/WI.

This bit always reads as zero.

`0b0` Send capture event to Secure monitors only.

`0b1` Send capture event to both Secure and Non-secure monitors.

NOW, bit [0]

When written as 1, this bit causes an event to all monitors in this MSC with CAPT_EVNT set to the value of 7.

When this bit is written as 0, no event is signaled.

This bit always reads as zero.

Accessing the MSMON_CAPT_EVNT:

This register is within the MPAM feature page memory frames. In a system that supports Secure and Non-secure memory maps, there must be both Secure and Non-secure MPAM feature pages.

MSMON_CAPT_EVNT_s must be accessible from the Secure MPAM feature page. MSMON_CAPT_EVENT_ns must be accessible from the Non-secure MPAM feature page.

The two instances of MSMON_CAPT_EVNT must be separate registers. The Secure instance (MSMON_CAPT_EVNT_s) can generate capture events for both Secure and Non-secure PARTID monitors, and the Non-secure instance (MSMON_CAPT_EVNT_ns) can generate capture events for Non-secure PARTID monitors only.

MSMON_CAPT_EVNT can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_s	0x0808	MSMON_CAPT_EVNT_s

This interface is accessible as follows:

- Accesses to this register are RW.

MSMON_CAPT_EVNT can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_ns	0x0808	MSMON_CAPT_EVNT_ns

This interface is accessible as follows:

- Accesses to this register are RW.

11.5.2 MSMON_CFG_CSU_CTL, MPAM Memory System Monitor Configure Cache Storage Usage Monitor Control Register

The MSMON_CFG_CSU_CTL characteristics are:

Purpose

MSMON_CFG_CSU_CTL is a 32-bit read-write register that controls the CSU monitor selected by MSMON_CFG_MON_SEL. MSMON_CFG_CSU_CTL_s controls the Secure cache storage usage monitor instance selected by the Secure instance of MSMON_CFG_MON_SEL. MSMON_CFG_CSU_CTL_ns controls Non-secure cache storage usage monitor instance selected by the Non-secure instance of MSMON_CFG_MON_SEL.

Configurations

The power domain of MSMON_CFG_CSU_CTL is IMPLEMENTATION DEFINED.

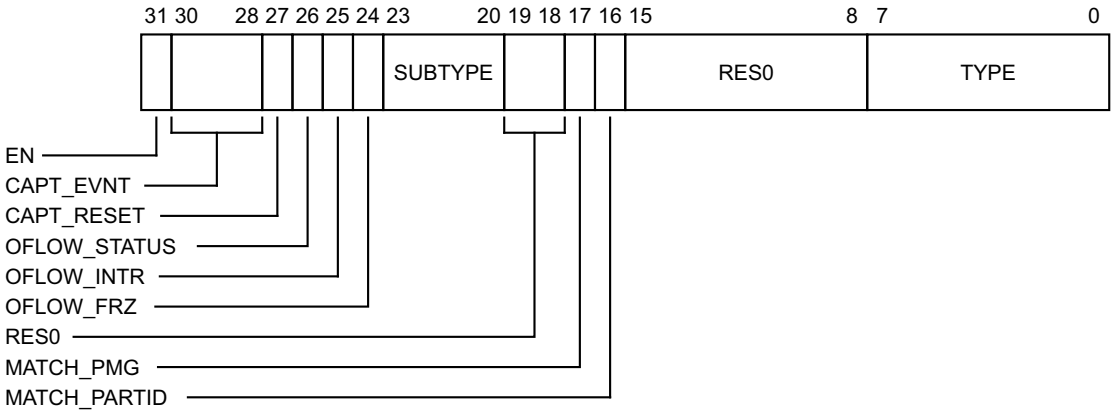
This register is present only when MPAMF_IDR.HAS_MSMON == 1 and MPAMF_MSMON_IDR.MSMON_CSU == 1. Otherwise, direct accesses to MSMON_CFG_CSU_CTL are RES0.

Attributes

MSMON_CFG_CSU_CTL is a 32-bit register.

Field descriptions

The MSMON_CFG_CSU_CTL bit assignments are:



EN, bit [31]

Enabled.

`0b0` The monitor is disabled and must not collect any information.

`0b1` The monitor is enabled to collect information according to its configuration.

CAPT_EVNT, bits [30:28]

Capture event selector.

Select the event that triggers capture from the following:

`0b000` No capture event is triggered.

`0b001` External capture event 1 (optional but recommended)

`0b010` External capture event 2 (optional)

`0b011` External capture event 3 (optional)

`0b100` External capture event 4 (optional)

`0b101` External capture event 5 (optional)

`0b110` External capture event 6 (optional)

`0b111` Capture occurs when a MSMON_CAPT_EVNT register in this MSC is written and causes a capture event for the security state of this monitor. (optional)

The values marked as optional indicate capture event sources that can be omitted in an implementation. Those values representing non-implemented event sources should not trigger a capture event.

If capture is not implemented for the CSU monitor type as indicated by MPAMF_CSUMON_IDR.HAS_CAPTURE = 0, this field is RAZ/WI.

CAPT_RESET, bit [27]

Reset after capture.

Controls whether the value of MSMON_CSU is reset to zero immediately after being copied to MSMON_CSU_CAPTURE.

`0b0` Monitor is not reset on capture.

`0b1` Monitor is reset on capture.

If capture is not implemented for the CSU monitor type as indicated by MPAMF_CSUMON_IDR.HAS_CAPTURE = 0, this field is RAZ/WI.

Because the CSU monitor type produces a measurement rather than a count, it might not make sense to ever reset the value after a capture. If there is no reason to ever reset a CSU monitor, this field is RAZ/WI.

OFLOW_STATUS, bit [26]

Overflow status.

Indicates whether the value of MSMON_CSU has overflowed.

`0b0` No overflow has occurred.

`0b1` At least one overflow has occurred since this bit was last written to zero.

If overflow is not possible for a CSU monitor in the implementation, this field is RAZ/WI.

OFLOW_INTR, bit [25]

Overflow Interrupt.

Indicates whether the value of MSMON_CSU has overflowed.

`0b0` No interrupt is signaled on an overflow of MSMON_CSU.

`0b1` On overflow, an implementation-specific interrupt is signaled.

If OFLOW_INTR is not supported by the implementation, this field is RAZ/WI.

OFLOW_FRZ, bit [24]

Freeze Monitor on Overflow.

Controls whether the value of MSMON_CSU freezes on an overflow.

`0b0` Monitor count wraps on overflow.

`0b1` Monitor count freezes on overflow. The frozen value might be 0 or another value if the monitor overflowed with an increment larger than 1.

If overflow is not possible for a CSU monitor in the implementation, this field is RAZ/WI.

SUBTYPE, bits [23:20]

Subtype.

A monitor can have other event matching criteria.

This field is not currently used for CSU monitors, but reserved for future use.

This field is RAZ/WI.

Bits [19:18]

Reserved, RES0.

MATCH_PMG, bit [17]

Match PMG.

Controls whether the monitor measures only storage used with PMG matching MSMON_CFG_CSU_FLT.PMG.

`0b0` The monitor measures storage used with any PMG value.

`0b1` The monitor only measures storage used with the PMG value matching MSMON_CFG_CSU_FLT.PMG.

If MATCH_PMG == 1 and MATCH_PARTID == 0, it is CONSTRAINED UNPREDICTABLE whether the monitor instance:

- Measures the storage used with matching PMG and with any PARTID.
- Measures no storage usage, that is, the monitor's VALUE field is zero.
- Measures the storage used with matching PMG and PARTID, that is, treats MATCH_PARTID as == 1.

MATCH_PARTID, bit [16]

Match PARTID.

Controls whether the monitor measures only storage used with PARTID matching MSMON_CFG_CSU_FLT.PARTID.

`0b0` The monitor measures storage used with any PARTID value.

`0b1` The monitor only measures storage used with the PARTID value matching MSMON_CFG_CSU_FLT.PARTID.

Bits [15:8]

Reserved, RES0.

TYPE, bits [7:0]

Monitor Type Code.

Constant type indicating the type of the monitor.

Read-only.

CSU monitor is TYPE = `0x43`.

Accessing the MSMON_CFG_CSU_CTL:

This register is within the MPAM feature page memory frames. In a system that supports Secure and Non-secure memory maps, there must be both Secure and Non-secure MPAM feature pages.

MSMON_CFG_CSU_CTL_s must be accessible from the Secure MPAM feature page.
MSMON_CFG_CSU_CTL_ns must be accessible from the Non-secure MPAM feature page.

MSMON_CFG_CSU_CTL_s and MSMON_CFG_CSU_CTL_ns must be separate registers. The Secure instance (MSMON_CFG_CSU_CTL_s) accesses the cache storage usage monitor controls used for Secure PARTIDs, and the Non-secure instance (MSMON_CFG_CSU_CTL_ns) accesses the cache storage usage monitor controls used for Non-secure PARTIDs.

MSMON_CFG_CSU_CTL can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_s	`0x0818`	MSMON_CFG_CSU_CTL_s

This interface is accessible as follows:

- Accesses to this register are RW.

MSMON_CFG_CSU_CTL can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_ns	0x0818	MSMON_CFG_CSU_CTL_ns

This interface is accessible as follows:

- Accesses to this register are RW.

11.5.3 MSMON_CFG_CSU_FLT, MPAM Memory System Monitor Configure Cache Storage Usage Monitor Filter Register

The MSMON_CFG_CSU_FLT characteristics are:

Purpose

MSMON_CFG_CSU_FLT is a 32-bit read-write register that sets PARTID and PMG to measure or count in the CSU monitor selected by MSMON_CFG_MON_SEL. MSMON_CFG_CSU_FLT_s sets filter conditions for the Secure cache storage usage monitor instance selected by the Secure instance of MSMON_CFG_MON_SEL. MSMON_CFG_CSU_CTL_ns sets filter conditions for the Non-secure cache storage usage monitor instance selected by the Non-secure instance of MSMON_CFG_MON_SEL.

Configurations

The power domain of MSMON_CFG_CSU_FLT is IMPLEMENTATION DEFINED.

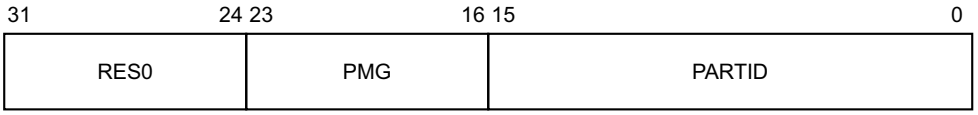
This register is present only when MPAMF_IDR.HAS_MSMON == 1 and MPAMF_MSMON_IDR.MSMON_CSU == 1. Otherwise, direct accesses to MSMON_CFG_CSU_FLT are RES0.

Attributes

MSMON_CFG_CSU_FLT is a 32-bit register.

Field descriptions

The MSMON_CFG_CSU_FLT bit assignments are:



Bits [31:24]

Reserved, RES0.

PMG, bits [23:16]

Performance monitoring group to filter cache storage usage monitoring.

If MSMON_CFG_CSU_CTL.MATCH_PMG == 0, this field is not used to match cache storage to a PMG and the contents of this field is ignored.

If MSMON_CFG_CSU_CTL.MATCH_PMG == 1 and MSMON_CFG_CSU_CTL.MATCH_PARTID == 1, the monitor instance selected by MSMON_CFG_MON_SEL measures or counts cache storage labeled with PMG equal to this field and PARTID equal to the PARTID field.

If MSMON_CFG_CSU_CTL.MATCH_PMG == 1 and MSMON_CFG_CSU_CTL.MATCH_PARTID == 0, the behavior of the monitor instance selected by MSMON_CFG_MON_SEL is CONSTRAINED UNPREDICTABLE. See MSMON_CFG_CSU_CTL.MATCH_PMG for more information.

PARTID, bits [15:0]

Partition ID to filter cache storage usage monitoring.

If MSMON_CFG_CSU_CTL.MATCH_PARTID == 0 and MSMON_CFG_CSU_CTL.MATCH_PMG == 0, the monitor measures all allocated cache storage.

If MSMON_CFG_CSU_CTL.MATCH_PARTID == 0 and MSMON_CFG_CSU_CTL.MATCH_PMG == 1, the monitor's behavior is CONSTRAINED UNPREDICTABLE. See the description of MSMON_CFG_CSU_CTL.MATCH_PMG.

If MSMON_CFG_CSU_CTL.MATCH_PARTID == 1 and MSMON_CFG_CSU_CTL.MATCH_PMG == 0, the monitor selected by MSMON_CFG_MON_SEL measures or counts cache storage labeled with PARTID equal to this field.

If MSMON_CFG_CSU_CTL.MATCH_PARTID == 1 and MSMON_CFG_CSU_CTL.MATCH_PMG == 1, the monitor selected by MSMON_CFG_MON_SEL measures or counts cache storage labeled with PARTID equal to this field and PMG equal to the PMG field.

Accessing the MSMON_CFG_CSU_FLT:

This register is within the MPAM feature page memory frames. In a system that supports Secure and Non-secure memory maps, there must be both Secure and Non-secure MPAM feature pages.

MSMON_CFG_CSU_FLT_s must be accessible from the Secure MPAM feature page.
MSMON_CFG_CSU_FLT_ns must be accessible from the Non-secure MPAM feature page.

MSMON_CFG_CSU_FLT_s and MSMON_CFG_CSU_FLT_ns must be separate registers. The Secure instance (MSMON_CFG_CSU_FLT_s) accesses the PARTID and PMG matching for a cache storage usage monitor used for Secure PARTIDs, and the Non-secure instance (MSMON_CFG_CSU_FLT_ns) accesses the PARTID and PMG matching for a cache storage usage monitor used for Non-secure PARTIDs.

MSMON_CFG_CSU_FLT can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_s	0x0810	MSMON_CFG_CSU_FLT_s

This interface is accessible as follows:

- Accesses to this register are RW.

MSMON_CFG_CSU_FLT can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_ns	0x0810	MSMON_CFG_CSU_FLT_ns

This interface is accessible as follows:

- Accesses to this register are RW.

11.5.4 MSMON_CFG_MBWU_CTL, MPAM Memory System Monitor Configure Memory Bandwidth Usage Monitor Control Register

The MSMON_CFG_MBWU_CTL characteristics are:

Purpose

MSMON_CFG_MBWU_CTL is a 32-bit read-write register that controls the MBWU monitor selected by MSMON_CFG_MON_SEL. MSMON_CFG_MBWU_CTL_s controls the Secure memory bandwidth usage monitor instance selected by the Secure instance of MSMON_CFG_MON_SEL. MSMON_CFG_MBWU_CTL_ns controls Non-secure memory bandwidth usage monitor instance selected by the Non-secure instance of MSMON_CFG_MON_SEL.

Configurations

The power domain of MSMON_CFG_MBWU_CTL is IMPLEMENTATION DEFINED.

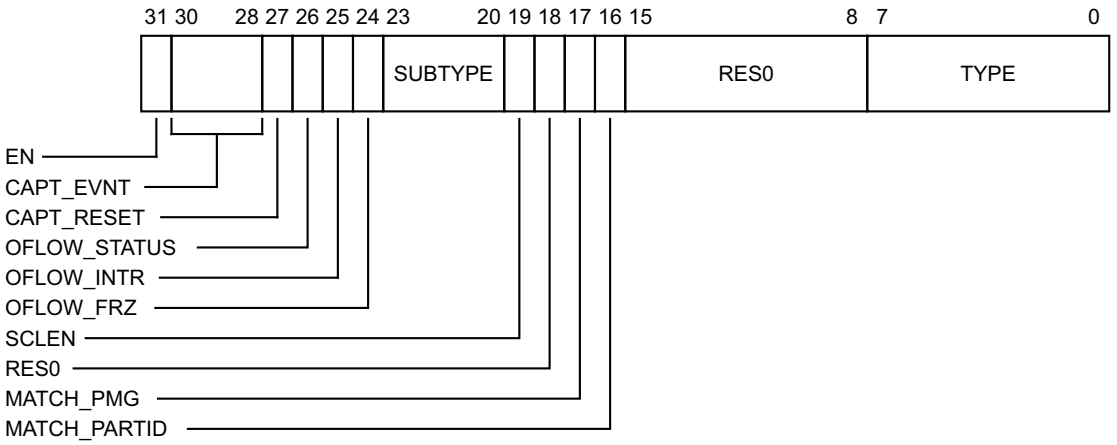
This register is present only when MPAMF_IDR.HAS_MSMON == 1 and MPAMF_MSMON_IDR.MSMON_MBWU == 1. Otherwise, direct accesses to MSMON_CFG_MBWU_CTL are RES0.

Attributes

MSMON_CFG_MBWU_CTL is a 32-bit register.

Field descriptions

The MSMON_CFG_MBWU_CTL bit assignments are:



EN, bit [31]

Enabled.

`0b0` The monitor is disabled and must not collect any information.

`0b1` The monitor is enabled to collect information according to its configuration.

CAPT_EVNT, bits [30:28]

Capture event selector.

Select the event that triggers capture from the following:

`0b000` No capture event is triggered.

`0b001` External capture event 1 (optional but recommended)

`0b010` External capture event 2 (optional)

`0b011` External capture event 3 (optional)

`0b100` External capture event 4 (optional)

`0b101` External capture event 5 (optional)

`0b110` External capture event 6 (optional)

`0b111` Capture occurs when a MSMON_CAPT_EVNT register in this MSC is written and causes a capture event for the security state of this monitor. (optional)

The values marked as optional indicate capture event sources that can be omitted in an implementation. Those values representing non-implemented event sources should not trigger a capture event.

If capture is not implemented for the MBWU monitor type as indicated by MPAMF_MBWUMON_IDR.HAS_CAPTURE = 0, this field is RAZ/WI.

CAPT_RESET, bit [27]

Reset after capture.

Controls whether the value of MSMON_MBWU is reset to zero immediately after being copied to MSMON_MBWU_CAPTURE.

`0b0` Monitor is not reset on capture.

`0b1` Monitor is reset on capture.

If capture is not implemented for the MBWU monitor type as indicated by MPAMF_MBWUMON_IDR.HAS_CAPTURE = 0, this field is RAZ/WI.

OFLOW_STATUS, bit [26]

Overflow status.

Indicates whether the value of MSMON_MBWU has overflowed.

`0b0` No overflow has occurred.

`0b1` At least one overflow has occurred since this bit was last written to zero.

If overflow is not possible for a MBWU monitor in the implementation, this field is RAZ/WI.

OFLOW_INTR, bit [25]

Overflow Interrupt.

Indicates whether an MPAM overflow interrupt is generated when the value of MSMON_MBWU overflows.

`0b0` No interrupt is signaled on an overflow of MSMON_MBWU.

`0b1` On overflow, an implementation-specific interrupt is signaled.

If OFLOW_INTR is not supported by the implementation, this field is RAZ/WI.

OFLOW_FRZ, bit [24]

Freeze Monitor on Overflow.

Controls whether the value of MSMON_MBWU freezes on an overflow.

`0b0` Monitor count wraps on overflow.

`0b1` Monitor count freezes on overflow. The frozen value might be 0 or another value if the monitor overflowed with an increment larger than 1.

If overflow is not possible for a MBWU monitor in the implementation, this field is RAZ/WI.

SUBTYPE, bits [23:20]

Subtype.

A monitor can have other event matching criteria.

This field is not currently used for MBWU monitors, but reserved for future use.

This field is RAZ/WI.

SCLEN, bit [19]

VALUE Scaling Enable.

Enables scaling of MSMON_MBWU by MPAMF_MBWUMON_IDR.SCALE.

`0b0` The monitor's VALUE field has bytes counted by the monitor.

`0b1` The monitor's VALUE field has bytes counted, shifted right by MPAMF_MBWUMON_IDR.SCALE.

Bit [18]

Reserved, RES0.

MATCH_PMG, bit [17]

Match PMG.

Controls whether the monitor measures only data transferred with PMG matching MSMON_CFG_MBWU_FLT.PMG.

`0b0` The monitor measures data transferred with any PMG value.

`0b1` The monitor only measures data transferred with the PMG value matching MSMON_CFG_MBWU_FLT.PMG.

MATCH_PARTID, bit [16]

Match PARTID.

Controls whether the monitor measures only data transferred with PARTID matching MSMON_CFG_MBWU_FLT.PARTID.

`0b0` The monitor measures data transferred with any PARTID value.

`0b1` The monitor only measures data transferred with the PARTID value matching MSMON_CFG_MBWU_FLT.PARTID.

Bits [15:8]

Reserved, RES0.

TYPE, bits [7:0]

Monitor Type Code.

Constant type indicating the type of the monitor.

Read-only.

MBWU monitor is TYPE = `0x42`.

Accessing the MSMON_CFG_MBWU_CTL:

This register is within the MPAM feature page memory frames. In a system that supports Secure and Non-secure memory maps, there must be both Secure and Non-secure MPAM feature pages.

MSMON_CFG_MBWU_CTL_s must be accessible from the Secure MPAM feature page.
MSMON_CFG_MBWU_CTL_ns must be accessible from the Non-secure MPAM feature page.

MSMON_CFG_MBWU_CTL_s and MSMON_CFG_MBWU_CTL_ns must be separate registers. The Secure instance (MSMON_CFG_MBWU_CTL_s) accesses the memory bandwidth usage monitor controls used for Secure PARTIDs, and the Non-secure instance (MSMON_CFG_MBWU_CTL_ns) accesses the memory bandwidth usage monitor controls used for Non-secure PARTIDs.

MSMON_CFG_MBWU_CTL can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_s	`0x0828`	MSMON_CFG_MBWU_CTL_s

This interface is accessible as follows:

- Accesses to this register are RW.

MSMON_CFG_MBWU_CTL can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_ns	0x0828	MSMON_CFG_MBWU_CTL_ns

This interface is accessible as follows:

- Accesses to this register are RW.

11.5.5 MSMON_CFG_MBWU_FLT, MPAM Memory System Monitor Configure Memory Bandwidth Usage Monitor Filter Register

The MSMON_CFG_MBWU_FLT characteristics are:

Purpose

MSMON_CFG_MBWU_FLT is a 32-bit read-write register that sets PARTID and PMG to measure or count in the MBWU monitor selected by MSMON_CFG_MON_SEL.
MSMON_CFG_MBWU_FLT_s sets filter conditions for the Secure memory bandwidth usage monitor instance selected by the Secure instance of MSMON_CFG_MON_SEL.
MSMON_CFG_MBWU_CTL_ns sets filter conditions for the Non-secure memory bandwidth usage monitor instance selected by the Non-secure instance of MSMON_CFG_MON_SEL.

Configurations

The power domain of MSMON_CFG_MBWU_FLT is IMPLEMENTATION DEFINED.

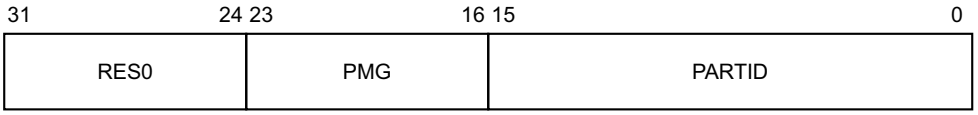
This register is present only when MPAMF_IDR.HAS_MSMON == 1 and MPAMF_MSMON_IDR.MSMON_MBWU == 1. Otherwise, direct accesses to MSMON_CFG_MBWU_FLT are RES0.

Attributes

MSMON_CFG_MBWU_FLT is a 32-bit register.

Field descriptions

The MSMON_CFG_MBWU_FLT bit assignments are:



Bits [31:24]

Reserved, RES0.

PMG, bits [23:16]

Performance monitoring group to filter memory bandwidth usage monitoring.

If MSMON_CFG_MBWU_CTL.MATCH_PMG == 0, this field is not used to match memory bandwidth to a PMG and the contents of this field is ignored.

If MSMON_CFG_MBWU_CTL.MATCH_PMG == 1, the monitor selected by MSMON_CFG_MON_SEL measures or counts memory bandwidth labeled with PMG equal to this field.

PARTID, bits [15:0]

Partition ID to filter memory bandwidth usage monitoring.

If MSMON_CFG_MBWU_CTL.MATCH_PARTID == 0, this field is not used to match memory bandwidth to a PARTID and the contents of this field is ignored.

If MSMON_CFG_MBWU_CTL.MATCH_PARTID == 1, the monitor selected by MSMON_CFG_MON_SEL measures or counts memory bandwidth labeled with PARTID equal to this field.

Accessing the MSMON_CFG_MBWU_FLT:

This register is within the MPAM feature page memory frames. In a system that supports Secure and Non-secure memory maps, there must be both Secure and Non-secure MPAM feature pages.

MSMON_CFG_MBWU_FLT_s must be accessible from the Secure MPAM feature page.
MSMON_CFG_MBWU_FLT_ns must be accessible from the Non-secure MPAM feature page.

MSMON_CFG_MBWU_FLT_s and MSMON_CFG_MBWU_FLT_ns must be separate registers. The Secure instance (MSMON_CFG_MBWU_FLT_s) accesses the PARTID and PMG matching for a memory bandwidth usage monitor used for Secure PARTIDs, and the Non-secure instance (MSMON_CFG_MBWU_FLT_ns) accesses the PARTID and PMG matching for a memory bandwidth usage monitor used for Non-secure PARTIDs.

MSMON_CFG_MBWU_FLT can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_s	0x0820	MSMON_CFG_MBWU_FLT_s

This interface is accessible as follows:

- Accesses to this register are RW.

MSMON_CFG_MBWU_FLT can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_ns	0x0820	MSMON_CFG_MBWU_FLT_ns

This interface is accessible as follows:

- Accesses to this register are RW.

11.5.6 MSMON_CFG_MON_SEL, MPAM Monitor Instance Selection Register

The MSMON_CFG_MON_SEL characteristics are:

Purpose

Selects a monitor instance to access through the MSMON configuration and counter registers. MSMON_CFG_MON_SEL_s selects a Secure monitor instance to access via the Secure MPAM feature page. MSMON_CFG_MON_SEL_ns selects a Non-secure monitor instance to access via the Non-secure MPAM feature page.

Note

Different performance monitoring features within a MSC could have different numbers of monitor instances. See the feature's NUM_MON field in its ID register. Thus, a monitor out-of-bounds error might be signaled when an MSMON_CFG register is accessed because the value in MSMON_CFG_MON_SEL.MON_SEL is too large for the particular monitoring feature.

To configure a monitor, set MON_SEL in this register to the index of the monitor instance to configure, then write to the MSMON_CFG_x register to set the configuration of the monitor. At a later time, read the monitor register (for example MSMON_MBWU) to get the value of the monitor.

Configurations

The power domain of MSMON_CFG_MON_SEL is IMPLEMENTATION DEFINED.

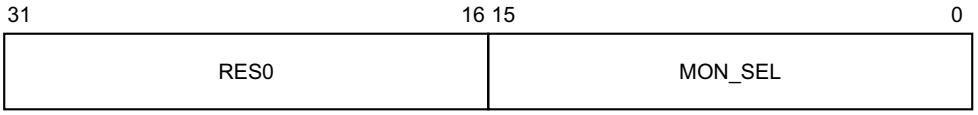
This register is present only when MPAMF_IDR.HAS_MSMON == 1 or MPAMF_IDR.HAS_IMPL_IDR == 1. Otherwise, direct accesses to MSMON_CFG_MON_SEL are RES0.

Attributes

MSMON_CFG_MON_SEL is a 32-bit register.

Field descriptions

The MSMON_CFG_MON_SEL bit assignments are:



Bits [31:16]

Reserved, RES0.

MON_SEL, bits [15:0]

Selects the monitor instance to configure or read.

Reads and writes to other MSMON registers are indexed by MON_SEL and by the NS bit used to access MSMON_CFG_MON_SEL to access the configuration for a single monitor.

Accessing the MSMON_CFG_MON_SEL:

This register is part of the MPAMF_BASE memory frame. In a system that supports Secure and Non-secure memory maps, the MPAMF_BASE frame must be accessible in both Secure and Non-secure memory address maps.

MSMON_CFG_MON_SEL must be accessible from the Non-secure and Secure address maps.

MSMON_CFG_MON_SEL must be banked for the Secure and Non-secure address maps. The Secure instance is used with accesses to other MSMON registers to configure monitors for Secure PARTIDs, and the Non-secure instance is used with accesses to other MSMON registers to configure monitors for Non-secure PARTIDs.

MSMON_CFG_MON_SEL can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_s	0x0800	MSMON_CFG_MON_SEL_s

This interface is accessible as follows:

- Accesses to this register are RW.

MSMON_CFG_MON_SEL can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_ns	0x0800	MSMON_CFG_MON_SEL_ns

This interface is accessible as follows:

- Accesses to this register are RW.

11.5.7 MSMON_CSU, MPAM Cache Storage Usage Monitor Register

The MSMON_CSU characteristics are:

Purpose

MSMON_CSU is a 32-bit read-write register that accesses the CSU monitor selected by MSMON_CFG_MON_SEL. MSMON_CSU_s is the Secure cache storage usage monitor instance selected by the Secure instance of MSMON_CFG_MON_SEL. MSMON_CSU_ns is the Non-secure cache storage usage monitor instance selected by the Non-secure instance of MSMON_CFG_MON_SEL.

Configurations

The power domain of MSMON_CSU is IMPLEMENTATION DEFINED.

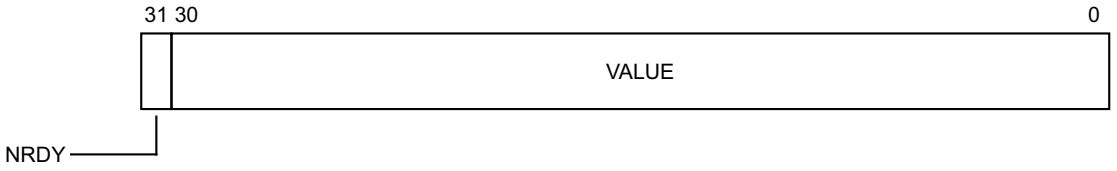
This register is present only when MPAMF_IDR.HAS_MSMON == 1 and MPAMF_MSMON_IDR.MSMON_CSU == 1. Otherwise, direct accesses to MSMON_CSU are RES0.

Attributes

MSMON_CSU is a 32-bit register.

Field descriptions

The MSMON_CSU bit assignments are:



NRDY, bit [31]

Not Ready. Indicates whether the monitor has possibly inaccurate data.

`0b0` The monitor is not ready and the contents of the VALUE field might be inaccurate or otherwise not represent the actual cache storage usage.

`0b1` The monitor is ready and the VALUE fields is accurate.

VALUE, bits [30:0]

Cache storage usage value if NRDY == 0. Invalid if NRDY == 1.

VALUE is the cache storage usage in bytes meeting the criteria set in MSMON_CFG_CSU_FLT and MSMON_CFG_CSU_CTL for the monitor instance selected by MSMON_CFG_MON_SEL.

Accessing the MSMON_CSU:

This register is within the MPAM feature page memory frames. In a system that supports Secure and Non-secure memory maps, there must be both Secure and Non-secure MPAM feature pages.

MSMON_CSU_s must be accessible from the Secure MPAM feature page. MSMON_CSU_ns must be accessible from the Non-secure MPAM feature page.

MSMON_CSU_s and MSMON_CSU_ns must be separate registers. The Secure instance (MSMON_CSU_s) accesses the cache storage usage monitor used for Secure PARTIDs, and the Non-secure instance (MSMON_CSU_ns) accesses the cache storage usage monitor used for Non-secure PARTIDs.

MSMON_CSU can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_s	0x0840	MSMON_CSU_s

This interface is accessible as follows:

- Accesses to this register are RW.

MSMON_CSU can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_ns	0x0840	MSMON_CSU_ns

This interface is accessible as follows:

- Accesses to this register are RW.

11.5.8 MSMON_CSU_CAPTURE, MPAM Cache Storage Usage Monitor Capture Register

The MSMON_CSU_CAPTURE characteristics are:

Purpose

MSMON_CSU_CAPTURE is a 32-bit read-write register that accesses the captured MSMON_CSU monitor selected by MSMON_CFG_MON_SEL. MSMON_CSU_CAPTURE_s is the Secure cache storage usage monitor capture instance selected by the Secure instance of MSMON_CFG_MON_SEL. MSMON_CSU_CAPTURE_ns is the Non-secure cache storage usage monitor capture instance selected by the Non-secure instance of MSMON_CFG_MON_SEL.

Configurations

The power domain of MSMON_CSU_CAPTURE is IMPLEMENTATION DEFINED.

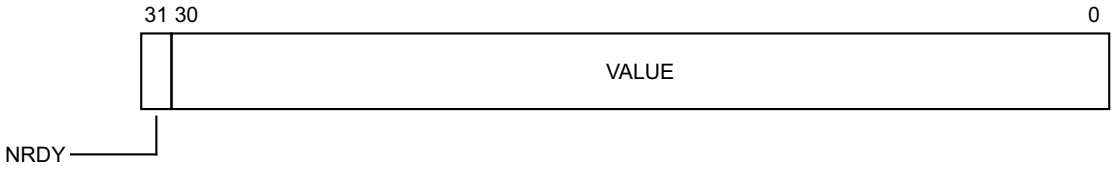
This register is present only when MPAMF_IDR.HAS_MSMON == 1, MPAMF_MSMON_IDR.MSMON_CSU == 1 and MPAMF_CSUMON_IDR.HAS_CAPTURE == 1. Otherwise, direct accesses to MSMON_CSU_CAPTURE are RES0.

Attributes

MSMON_CSU_CAPTURE is a 32-bit register.

Field descriptions

The MSMON_CSU_CAPTURE bit assignments are:



NRDY, bit [31]

Not Ready. Indicates whether the captured monitor value has possibly inaccurate data.

`0b0` The captured monitor was not ready and the contents of the VALUE field might be inaccurate or otherwise not represent the actual cache storage usage.

`0b1` The captured monitor was ready and the VALUE fields is accurate.

VALUE, bits [30:0]

Captured cache storage usage value if NRDY == 0. Invalid if NRDY == 1.

VALUE is the captured cache storage usage in bytes meeting the criteria set in MSMON_CFG_CSU_FLT and MSMON_CFG_CSU_CTL for the monitor instance selected by MSMON_CFG_MON_SEL.

Accessing the MSMON_CSU_CAPTURE:

This register is within the MPAM feature page memory frames. In a system that supports Secure and Non-secure memory maps, there must be both Secure and Non-secure MPAM feature pages.

MSMON_CSU_CAPTURE_s must be accessible from the Secure MPAM feature page. MSMON_CSU_CAPTURE_ns must be accessible from the Non-secure MPAM feature page.

MSMON_CSU_CAPTURE_s and MSMON_CSU_CAPTURE_ns must be separate registers. The Secure instance (MSMON_CSU_CAPTURE_s) accesses the captured cache storage usage monitor used for Secure PARTIDs, and the Non-secure instance (MSMON_CSU_CAPTURE_ns) accesses the captured cache storage usage monitor used for Non-secure PARTIDs.

MSMON_CSU_CAPTURE can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_s	0x0848	MSMON_CSU_CAPTURE_s

This interface is accessible as follows:

- Accesses to this register are RW.

MSMON_CSU_CAPTURE can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_ns	0x0848	MSMON_CSU_CAPTURE_ns

This interface is accessible as follows:

- Accesses to this register are RW.

11.5.9 MSMON_MBWU, MPAM Memory Bandwidth Usage Monitor Register

The MSMON_MBWU characteristics are:

Purpose

MSMON_MBWU is a 32-bit read-write register that accesses the monitor instance selected by MSMON_CFG_MON_SEL. MSMON_MBWU_s is the Secure memory bandwidth usage monitor instance selected by MSMON_CFG_MON_SEL_s. MSMON_MBWU_ns is the Non-secure memory bandwidth usage monitor instance selected by MSMON_CFG_MON_SEL_ns.

Configurations

The power domain of MSMON_MBWU is IMPLEMENTATION DEFINED.

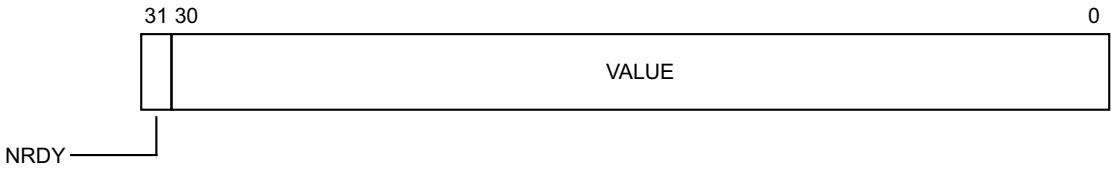
This register is present only when MPAMF_IDR.HAS_MSMON == 1 and MPAMF_MSMON_IDR.MSMON_MBWU == 1. Otherwise, direct accesses to MSMON_MBWU are RES0.

Attributes

MSMON_MBWU is a 32-bit register.

Field descriptions

The MSMON_MBWU bit assignments are:



NRDY, bit [31]

Not Ready. Indicates whether the monitor has possibly inaccurate data.

`0b0` The monitor is not ready and the contents of the VALUE field might be inaccurate or otherwise not represent the actual memory bandwidth usage.

`0b1` The monitor is ready and the VALUE fields is accurate.

VALUE, bits [30:0]

Memory bandwidth usage counter value if NRDY == 0. Invalid if NRDY == 1.

VALUE is the scaled count of bytes transferred since the monitor was last reset that meet the criteria set in MSMON_CFG_MBWU_FLT and MSMON_CFG_MBWU_CTL for the monitor instance selected by MSMON_CFG_MON_SEL.

If MSMON_CFG_MBWU_CTL.SCLEN == 1, the count in VALUE is the number of bytes shifted right by MPAMF_MBWUMON_IDR.SCALE bit positions and rounded. See section 10.2.1.

Accessing the MSMON_MBWU:

This register is within the MPAM feature page memory frames. In a system that supports Secure and Non-secure memory maps, there must be both Secure and Non-secure MPAM feature pages.

MSMON_MBWU_s must be accessible from the Secure MPAM feature page. MSMON_MBWU_ns must be accessible from the Non-secure MPAM feature page.

MSMON_MBWU_s and MSMON_MBWU_ns must be separate registers. The Secure instance (MSMON_MBWU_s) accesses the memory bandwidth usage monitor used for Secure PARTIDs, and the Non-secure instance (MSMON_MBWU_ns) accesses the memory bandwidth usage monitor used for Non-secure PARTIDs.

MSMON_MBWU can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_s	0x0860	MSMON_MBWU_s

This interface is accessible as follows:

- Accesses to this register are RW.

MSMON_MBWU can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_ns	0x0860	MSMON_MBWU_ns

This interface is accessible as follows:

- Accesses to this register are RW.

11.5.10 MSMON_MBWU_CAPTURE, MPAM Memory Bandwidth Usage Monitor Capture Register

The MSMON_MBWU_CAPTURE characteristics are:

Purpose

MSMON_MBWU_CAPTURE is a 32-bit read-write register that accesses the captured MSMON_MBWU monitor instance selected by MSMON_CFG_MON_SEL. MSMON_MBWU_CAPTURE_s is the Secure memory bandwidth usage monitor capture instance selected by the Secure instance of MSMON_CFG_MON_SEL. MSMON_MBWU_CAPTURE_ns is the Non-secure memory bandwidth usage monitor capture instance selected by the Non-secure instance of MSMON_CFG_MON_SEL.

Configurations

The power domain of MSMON_MBWU_CAPTURE is IMPLEMENTATION DEFINED.

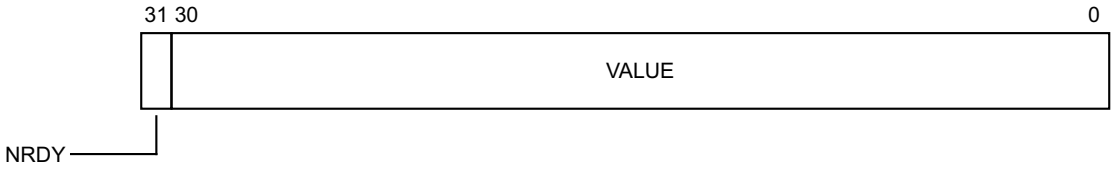
This register is present only when MPAMF_IDR.HAS_MSMON == 1, MPAMF_MSMON_IDR.MSMON_MBWU == 1 and MPAMF_MBWUMON_IDR.HAS_CAPTURE == 1. Otherwise, direct accesses to MSMON_MBWU_CAPTURE are RES0.

Attributes

MSMON_MBWU_CAPTURE is a 32-bit register.

Field descriptions

The MSMON_MBWU_CAPTURE bit assignments are:



NRDY, bit [31]

Not Ready. The captured NRDY bit from the corresponding instance of MSMON_MBWU. This bit indicates whether the captured monitor value has possibly inaccurate data.

`0b0` The captured monitor was not ready and the contents of the VALUE field might be inaccurate or otherwise not represent the actual memory bandwidth usage.

`0b1` The captured monitor was ready and the VALUE fields is accurate.

VALUE, bits [30:0]

Captured memory bandwidth usage counter value if NRDY == 0. Invalid if NRDY == 1.

VALUE is the captured VALUE field from the corresponding instance of MSMON_MBWU, the count of bytes transferred since the monitor was last reset that meet the criteria set in MSMON_CFG_MBWU_FLT and MSMON_CFG_MBWU_CTL for the monitor instance selected by MSMON_CFG_MON_SEL.

VALUE captures the MSMON_MBWU.VALUE and preserves any scaling that had been performed on the VALUE field in that register.

Accessing the MSMON_MBWU_CAPTURE:

This register is within the MPAM feature page memory frames. In a system that supports Secure and Non-secure memory maps, there must be both Secure and Non-secure MPAM feature pages.

MSMON_MBWU_CAPTURE_s must be accessible from the Secure MPAM feature page. MSMON_MBWU_CAPTURE_ns must be accessible from the Non-secure MPAM feature page.

MSMON_MBWU_CAPTURE_s and MSMON_MBWU_CAPTURE_ns must be separate registers. The Secure instance (MSMON_MBWU_CAPTURE_s) accesses the captured memory bandwidth usage monitor used for Secure PARTIDs, and the Non-secure instance (MSMON_MBWU_CAPTURE_ns) accesses the captured memory bandwidth usage monitor used for Non-secure PARTIDs.

MSMON_MBWU_CAPTURE can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_s	0x0868	MSMON_MBWU_CAPTURE_s

This interface is accessible as follows:

- Accesses to this register are RW.

MSMON_MBWU_CAPTURE can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_ns	0x0868	MSMON_MBWU_CAPTURE_ns

This interface is accessible as follows:

- Accesses to this register are RW.

11.6 Memory-mapped control and status registers

This section lists the external control and status registers.

11.6.1 MPAMF_ECR, MPAM Error Control Register

The MPAMF_ECR characteristics are:

Purpose

MPAMF_ECR is a 32-bit read-write register that controls MPAM error interrupts for this MSC. MPAMF_ECR_s controls Secure MPAM error handling. MPAMF_ECR_ns controls Non-secure MPAM error handling.

Configurations

The power domain of MPAMF_ECR is IMPLEMENTATION DEFINED.

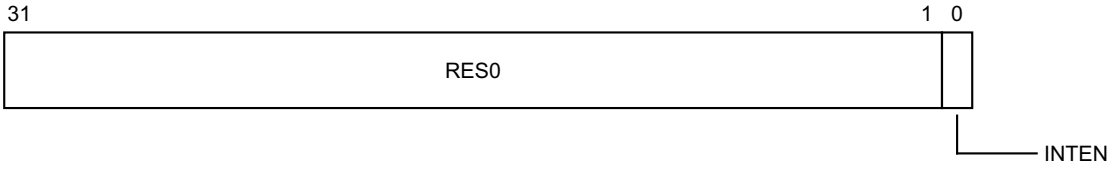
If a MSC cannot encounter any of the error conditions listed in section 15.1, both the MPAMF_ESR and MPAMF_ECR must be RAZ/WI.

Attributes

MPAMF_ECR is a 32-bit register.

Field descriptions

The MPAMF_ECR bit assignments are:



Bits [31:1]

Reserved, RES0.

INTEN, bit [0]

Interrupt Enable.

`0b0` MPAM error interrupts are not generated.

`0b1` MPAM error interrupts are generated.

Accessing the MPAMF_ECR:

This register is within the MPAM feature page memory frames. In a system that supports Secure and Non-secure memory maps, there must be both Secure and Non-secure MPAM feature pages.

MPAMF_ECR_s must be accessible from the Secure MPAM feature page. MPAMF_ECR_ns must be accessible from the Non-secure MPAM feature page.

MPAMF_ECR_s and MPAMF_ECR_ns must be separate registers. The Secure instance (MPAMF_ECR_s) accesses the error interrupt controls used for Secure PARTIDs, and the Non-secure instance (MPAMF_ECR_ns) accesses the error interrupt controls used for Non-secure PARTIDs.

MPAMF_ECR can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_s	`0x00F0`	MPAMF_ECR_s

This interface is accessible as follows:

- Accesses to this register are RW.

MPAMF_ECR can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_ns	0x00F0	MPAMF_ECR_ns

This interface is accessible as follows:

- Accesses to this register are RW.

11.6.2 MPAMF_ESR, MPAM Error Status Register

The MPAMF_ESR characteristics are:

Purpose

MPAMF_ESR is a 32-bit read-write register that gives MPAM error status for this MSC. MPAMF_ESR_s reports Secure MPAM errors. MPAMF_ESR_ns reports Non-secure MPAM errors.

Software should write this register after reading the status of an error to reset ERRCODE to `0x0000` and OVRWR to 0 so that future errors are not reported with OVRWR set.

Configurations

The power domain of MPAMF_ESR is IMPLEMENTATION DEFINED.

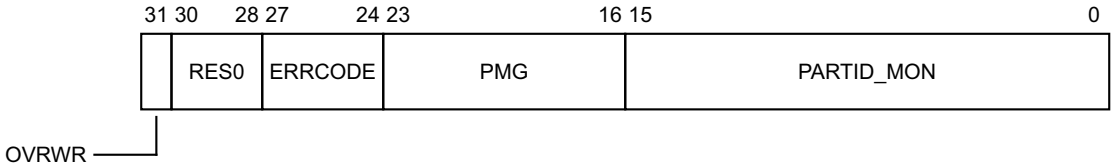
If a MSC cannot encounter any of the error conditions listed in section 15.1, both the MPAMF_ESR and MPAMF_ECR must be RAZ/WI.

Attributes

MPAMF_ESR is a 32-bit register.

Field descriptions

The MPAMF_ESR bit assignments are:



OVRWR, bit [31]

Overwritten.

If 0 and ERRCODE == `0b0000`, no errors have occurred.

If 0 and ERRCODE is non-zero, a single error has occurred and is recorded in this register.

If 1 and ERRCODE is non-zero, multiple errors have occurred and this register records the most recent error.

The state where this bit is 1 and ERRCODE is zero must not be produced by hardware and is only reached when software writes this combination into this register.

Bits [30:28]

Reserved, RES0.

ERRCODE, bits [27:24]

Error code. See section 15.1

`0b0000`	No error
`0b0001`	PARTID_SEL_Range
`0b0010`	Req_PARTID_Range
`0b0011`	MSMONCFG_ID_RANGE
`0b0100`	Req_PMG_Range
`0b0101`	Monitor_Range
`0b0110`	intPARTID_Range
`0b0111`	Unexpected_INTERNAL

0b1000	Reserved
0b1001	Reserved
0b1010	Reserved
0b1011	Reserved
0b1100	Reserved
0b1101	Reserved
0b1110	Reserved
0b1111	Reserved

PMG, bits [23:16]

Program monitoring group.

Set to the PMG on an error that captures PMG. Otherwise, set to 0x00 on an error that does not capture PMG.

PARTID_MON, bits [15:0]

PARTID or monitor.

Set to the PARTID on an error that captures PARTID.

Set to the monitor index on an error that captures MON.

On an error that captures neither PARTID nor MON, this field is set to 0x0000.

Accessing the MPAMF_ESR:

This register is within the MPAM feature page memory frames. In a system that supports Secure and Non-secure memory maps, there must be both Secure and Non-secure MPAM feature pages.

MPAMF_ESR_s must be accessible from the Secure MPAM feature page. MPAMF_ESR_ns must be accessible from the Non-secure MPAM feature page.

MPAMF_ESR_s and MPAMF_ESR_ns must be separate registers. The Secure instance (MPAMF_ESR_s) accesses the error status used for Secure PARTIDs, and the Non-secure instance (MPAMF_ESR_ns) accesses the error status used for Non-secure PARTIDs.

MPAMF_ESR can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_s	0x00F8	MPAMF_ESR_s

This interface is accessible as follows:

- Accesses to this register are RW.

MPAMF_ESR can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_ns	0x00F8	MPAMF_ESR_ns

This interface is accessible as follows:

- Accesses to this register are RW.

Chapter 12
Errors in MSCs

This chapter contains the following sections:

12.1 Introduction

When an MSC detects an error on an access to a memory-mapped register, information about the error must be captured in the MPAMF_ESR register and signaled to software via an interrupt. The errors covered by this mechanism could be caused by software errors.

Errors, whether detected or not, must not prevent the handling of the request by the MSC, but errors can cause the MSC to use different MPAM resource control settings than expected or cause monitors to mis-attribute monitored events.

Note

Implementation choices in an MSC may make certain errors impossible. For example, if the request interface only implements enough bits to exactly cover the range of 0 to PARTID_MAX, then the request PARTID cannot be detected as out-of-range, so ERRCODE == 2 could not occur.

MPAM errors that an implementation detects are recorded in MPAMF_ESR_s or MPAMF_ESR_ns. The error condition descriptions in *Error conditions in accessing memory-mapped registers* on page 12-243 describe whether the security state of the PARTID or of the request address are used to determine which instance of MPAMF_ESR records the error status.

MSCs signal errors in accesses to memory-mapped registers using an error interrupt. See *MPAM Error Interrupt* on page 8-124. Errors recorded in MPAMF_ESR_s signal a Secure MPAM error interrupt if enabled by MPAMF_ECR_s.INTEN == 1. Errors recorded in MPAM_ESR_ns signal a Non-secure MPAM error interrupt if enabled by MPAMF_ECR_ns.INTEN.

The MPAMF_ESR in an MSC captures the reason for an error, so that it can be accurately reported to software.

12.2 Error conditions in accessing memory-mapped registers

When an MSC detects an error condition, information about the error is captured in MPAMF_ESR. MPAMF_ESR.ERRCODE encodes the reason for the error as shown in Table 12-1. Other fields are captured in MPAMF_ESR as shown in the "Fields Captured" column of Table 12-1.

Table 12-1 Error conditions in accessing memory-mapped registers

MPAM Error Code (ERRCODE)	Error Name	Error Description	Fields Captured
0	No Error	No error captured in MPAMF_ESR.	None
1	PARTID_SEL_Range	MPAMCFG_PART_SEL stored with an out-of-range PARTID.	PARTID
2	Req_PARTID_Range	A request has out-of-range PARTID.	PARTID and PMG
3	MSMONCFG_ID_RANGE	MSMON configuration request has out-of-range PARTID or PMG.	PARTID and PMG
4	Req_PMG_Range	A request has out-of-range PMG.	PARTID and PMG
5	Monitor_Range	MSMON_CFG_MON_SEL has out-of-range monitor selector.	MON_SEL
6	intPARTID_Range	The intPARTID in MPAMCFG_INTPARTID is out of the intPARTID range for the PARTID in MPAMCFG_PART_SEL.	intPARTID
7	Unexpected_INTERNAL	MPAMCFG_PART_SEL.INTERNAL is set when a reqPARTID is expected.	PARTID
15:8	Reserved	Reserved for future use.	--

12.2.1 No error (errorcode == 0)

No error is captured in MPAMF_ESR.

12.2.2 PARTID_SEL out-of-range error (errorcode == 1)

The value of the MPAMCFG_PART_SEL.PARTID_SEL field is out-of-range for the PARTID space selected by the NS bit on a store to an MPAMCFG memory-mapped register.

The selection of the Secure or Non-secure version of MPAMF_ESR for capturing the error information is also controlled by the NS bit.

12.2.3 Request PARTID out-of-range error (errorcode == 2)

The value of PARTID in a request is out-of-range for the MSC's MPAM implementation of PARTID space selected by the MPAM_NS bit.

The selection of the Secure or Non-secure version of MPAMF_ESR for capturing the error information is also controlled by the MPAM_NS bit.

The MPAM behavior of an MSC for a request that causes this error is CONSTRAINED UNPREDICTABLE:

- The request may be processed as if the PARTID is any valid PARTID in the same MPAM Security state (MPAM_NS) as the request's PARTID.
- Arm recommends that the default PARTID for the MPAM_NS Security state is used. See *Default PARTID on page 3-30*.

12.2.4 MSMON configuration ID out-of-range error (errorcode == 3)

A write to configure a monitor contains an out-of-range value for either the PARTID or PMG for the PARTID space selected by the secure address space bit, NS.

The selection of the Secure or Non-secure version of MPAMF_ESR for capturing the error information is also controlled by the NS bit.

12.2.5 Request PMG out-of-range error (errorcode == 4)

The value of PMG in a request is out of range for the MSC's MPAM implementation of the PMG space selected by the MPAM security space bit, MPAM_NS.

The selection of the Secure or Non-secure version of MPAMF_ESR for capturing the error information is also controlled by the MPAM_NS bit.

The MPAM behavior of an MSC for a request that causes this error is CONSTRAINED UNPREDICTABLE:

- The request may be processed as if the PARTID and PMG are any valid PARTID and PMG in the same MPAM Security state as the request.
  - Arm recommends that the request be processed as if the PMG is the default. See *Default PARTID* on page 3-30.
- The default PARTID and PMG may be used for the request's MPAM_NS Security state. See *Default PARTID* on page 3-30. The request may be IGNORED for performance monitoring, as if the PMG value does not match the monitor's PMG filter even if the PARTID matches.

12.2.6 Monitor out-of-range error (errorcode == 5)

The value of the monitor selector register (MSMON_CFG_MON_SEL.MON_SEL) is out of range on a store to an MSMON_* memory-mapped register selected by the secure address space bit, NS.

The selection of the Secure or Non-secure version of MPAMF_ESR for capturing the error information is also controlled by the NS bit.

12.2.7 intPARTID out-of-range error (errorcode == 6)

This error can only occur if PARTID narrowing is implemented. MPAMF_IDR.HAS_PARTID_NRW == 1 indicates that an implementation has PARTID narrowing.

The selection of the Secure or Non-secure version of MPAMF_ESR for capturing the error information is controlled by the secure address space bit, NS.

These conditions cause this error:

- MPAMCFG_INTPARTID.INTPARTID is out-of-range for the intPARTID space selected by the secure address space bit, NS, on a store to a memory-mapped register to configure the association of reqPARTID to intPARTID.
- MPAMCFG_INTPARTID.INTERNAL == 0 on any write to configure MPAMCFG_INTPARTID.
- MPAMCFG_PART_SEL.INTERNAL is not set when an intPARTID is expected. These expected cases include a read or write to any MPAMCFG_* register, other than MPAMCFG_INTPARTID.

12.2.8 Unexpected INTERNAL error (errorcode == 7)

This error can only occur if PARTID narrowing is implemented. MPAMF_IDR.HAS_PARTID_NRW == 1 indicates that an implementation has PARTID narrowing.

If PARTID narrowing is implemented in the MSC, this error is detected if the MPAMCFG_PART_SEL.INTERNAL bit is set when a reqPARTID is expected. When PARTID narrowing is implemented, the only cases in which a reqPARTID is expected in MPAMCFG_PART_SEL are a read or write access to MPAMCFG_INTPARTID.

The selection of the Secure or Non-secure version of MPAMF_ESR for capturing the error information is controlled by the secure address space bit, NS.

Reads that cause this error return an UNKNOWN value.

12.2.9 Reserved (errcodes 8 – 15)

This error code is reserved for future use.

12.3 Overwritten error status

When MPAMF_ESR is written due to an error, and the ERRCODE field was not previously 0, the OVRWR bit is set. Error status is always written to MPAMF_ESR, whether or not it contains a previously recorded error syndrome.

Table 12-2 Overwritten error status

OVRWR	ERRCODE	Description
0	`0b0000`	No errors have been recorded in MPAMF_ESR.
0	Non-zero	Not overwritten. A single error has been written to MPAMF_ESR since it was last cleared.
1	`0b0000`	This state is not produced by hardware, only by a software write.
1	Non-zero	Overwritten. Two or more errors have been written to MPAMF_ESR with only the syndrome information from the latest error recorded into the fields.

The interrupt service routine should clear both the ERRCODE and OVRWR fields of MPAMF_ESR after its contents have been read. This allows the OVRWR bit to accurately indicate when one or more errors have been overwritten before servicing future MPAM error interrupts.

12.4 Behavior of configuration reads and writes with errors

A configuration read that has a detected error returns an UNKNOWN value.

If a configuration write to an MPAMCFG_* register has a PARTID_SEL out-of-range error (ERRCODE == 1), the control settings accessed are permitted to be for any PARTID or the write is permitted to be IGNORED. As a result, the control settings accessible from that MPAMCFG_* register for any valid PARTID might become UNKNOWN.

If there is no PARTID_SEL out-of-range error (ERRCODE == 1), a configuration write to an MPAMCFG_* register that has a detected error leaves the control settings for the partition selected by MPAMCFG_PART_SEL.PARTID_SEL in an UNKNOWN state.

If a configuration write to an MSMON register has a monitor out-of-range error (ERRCODE=5), the monitor settings accessed are permitted to be for any monitor or the write is permitted to be IGNORED. As a result, the control settings for any valid monitor might become UNKNOWN.

If there is no monitor out-of-range error (ERRCODE=5), a configuration write to an MSMON_* register that has a detected error leaves the settings for the monitor selected by MSMON_CFG_MON_SEL.MON_SEL in an UNKNOWN state.

Chapter 13
Armv8 Pseudocode

This chapter contains pseudocode that describes the generation of MPAM information by a PE following the MPAM architecture. It contains the following section:

13.1 Shared pseudocode

This section holds the pseudocode that is common to execution in AArch64 state and in AArch32 state. Functions listed in this section are identified only by a `FunctionName`, without an `AArch64.` or `AArch32.` prefix. This section is organized by functional groups, with the functional groups being indicated by hierarchical path names, for example `shared/functions/extension`.

The sections of the shared pseudocode hierarchy containing MPAM pseudocode are:

- *shared/functions/extension*.
- *shared/functions/memory*.
- *shared/functions/mpam* on page 13-251.

13.1.1 shared/functions/extension

This section includes the following pseudocode function:

- *shared/functions/extension/HaveMPAMExt*.

shared/functions/extension/HaveMPAMExt

```
// HaveMPAMExt()
// =============
// Returns TRUE if MPAM is implemented, and FALSE otherwise.

boolean HaveMPAMExt()
    return (HasArchVersion(ARMv8p2) &&
            boolean IMPLEMENTATION_DEFINED "Has MPAM extension");
```

13.1.2 shared/functions/memory

This section includes the following pseudocode functions:

- *shared/functions/memory/AccessDescriptor*.
- *shared/functions/memory/CreateAccessDescriptor*.
- *shared/functions/memory/CreateAccessDescriptorPTW* on page 13-251.
- *shared/functions/memory/MPAMinfo* on page 13-251.

shared/functions/memory/AccessDescriptor

```
type AccessDescriptor is (
    AccType acctype,
    MPAMinfo mpam,
    boolean page_table_walk,
    boolean secondstage,
    boolean s2fs1walk,
    integer level
)
```

shared/functions/memory/CreateAccessDescriptor

```
// CreateAccessDescriptor()
// ========================

AccessDescriptor CreateAccessDescriptor(AccType acctype)
    AccessDescriptor accdesc;
    accdesc.acctype = acctype;
    accdesc.mpam = GenMPAMcurEL(acctype IN {AccType_IFETCH, AccType_IC});
```

```
    accdesc.page_table_walk = FALSE;
    return accdesc;
```

shared/functions/memory/CreateAccessDescriptorPTW

```
// CreateAccessDescriptorPTW()
// ===========================

AccessDescriptor CreateAccessDescriptorPTW(AccType acctype, boolean secondstage,
                                           boolean s2fs1walk, integer level)
    AccessDescriptor accdesc;
    accdesc.acctype = acctype;
    accdesc.mpam = GenMPAMcurEL(acctype IN {AccType_IFETCH, AccType_IC});
    accdesc.page_table_walk = TRUE;
    accdesc.s2fs1walk = s2fs1walk;
    accdesc.secondstage = secondstage;
    accdesc.level = level;
    return accdesc;
```

shared/functions/memory/MPAMinfo

```
// MPAMinfo
// ========

// MPAM type definitions
type PARTIDtype = bits(16);
type PMGtype = bits(8);
type MPAMinfo is (
     bit mpam_ns,
     PARTIDtype partid,
     PMGtype pmg
)
```

13.1.3 shared/functions/mpam

This section includes the following pseudocode functions:

shared/functions/mpam/DefaultMPAMinfo

```
// DefaultMPAMinfo
// ===============
// Returns default MPAM info.  If secure is TRUE return default Secure
// MPAMinfo, otherwise return default Non-secure MPAMinfo.
```

```
MPAMinfo DefaultMPAMinfo(boolean secure)
    MPAMinfo DefaultInfo;
    DefaultInfo.mpam_ns = if secure then '0' else '1';
    DefaultInfo.partid  = DefaultPARTID;
    DefaultInfo.pmg     = DefaultPMG;
    return DefaultInfo;
```

shared/functions/mpam/DefaultPARTID

```
constant PARTIDtype DefaultPARTID = 0<15:0>;
```

shared/functions/mpam/DefaultPMG

```
constant PMGtype    DefaultPMG = 0<7:0>;
```

shared/functions/mpam/GenMPAMcurEL

```
// GenMPAMcurEL
// ============
// Returns MPAMinfo for the current EL and security state.
// InD is TRUE instruction access and FALSE otherwise.
// May be called if MPAM is not implemented (but in an version that supports
// MPAM), MPAM is disabled, or in AArch32.  In AArch32, convert the mode to
// EL if can and use that to drive MPAM information generation.  If mode
// cannot be converted, MPAM is not implemented, or MPAM is disabled return
// default MPAM information for the current security state.

MPAMinfo GenMPAMcurEL(boolean InD)
    bits(2) mpamel;
    boolean validEL;
    boolean secure = IsSecure();
    if HaveMPAMExt() && MPAMisEnabled() then
        if UsingAArch32() then
            (validEL, mpamel) = ELFromM32(PSTATE.M);
        else
            validEL = TRUE;
            mpamel = PSTATE.EL;
        if validEL then
            return genMPAM(UInt(mpamel), InD, secure);
    return DefaultMPAMinfo(secure);
```

shared/functions/mpam/MAP_vPARTID

```
// MAP_vPARTID
// ===========
// Performs conversion of virtual PARTID into physical PARTID
// Contains all of the error checking and implementation
// choices for the conversion.

(PARTIDtype, boolean) MAP_vPARTID(PARTIDtype vpartid)
    // should not ever be called if EL2 is not implemented
    // or is implemented but not enabled in the current
    // security state.
    PARTIDtype ret;
    boolean err;
    integer virt    = UInt( vpartid );
    integer vmprmax = UInt( MPAMIDR_EL1.VPMR_MAX );

    // vpartid_max is largest vpartid supported
    integer vpartid_max = 4 * vmprmax + 3;

    // One of many ways to reduce vpartid to value less than vpartid_max.
    if virt > vpartid_max then
        virt = virt MOD (vpartid_max+1);
```

```
    // Check for valid mapping entry.
    if MPAMVPMV_EL2<virt> == '1' then
        // vpartid has a valid mapping so access the map.
        ret = mapvpmw(virt);
        err = FALSE;

    // Is the default virtual PARTID valid?
    elsif MPAMVPMV_EL2<0> == '1' then
        // Yes, so use default mapping for vpartid == 0.
        ret = MPAMVPM0_EL2<0 +: 16>;
        err = FALSE;

    // Neither is valid so use default physical PARTID.
    else
        ret = DefaultPARTID;
        err = TRUE;

    // Check that the physical PARTID is in-range.
    // This physical PARTID came from a virtual mapping entry.
    integer partid_max = UInt( MPAMIDR_EL1.PARTID_MAX );
    if UInt(ret) > partid_max then
        // Out of range, so return default physical PARTID
        ret = DefaultPARTID;
        err = TRUE;
    return (ret, err);
```

shared/functions/mpam/MPAMisEnabled

```
// MPAMisEnabled
// =============
// Returns TRUE if MPAMisEnabled.

boolean MPAMisEnabled()
    el = HighestEL();
    case el of
        when EL3 return MPAM3_EL3.MPAMEN == '1';
        when EL2 return MPAM2_EL2.MPAMEN == '1';
        when EL1 return MPAM1_EL1.MPAMEN == '1';
```

shared/functions/mpam/MPAMisVirtual

```
// MPAMisVirtual
// =============
// Returns TRUE if MPAM is configured to be virtual at EL.

boolean MPAMisVirtual(integer el)
    return ( MPAMIDR_EL1.HAS_HCR == '1' && EL2Enabled() &&
             ( HCR_EL2.E2H == '0' || HCR_EL2.TGE == '0' ) &&
             (( el == 0 && MPAMHCR_EL2.EL0_VPMEN == '1' ) ||
              ( el == 1 && MPAMHCR_EL2.EL1_VPMEN == '1')));
```

shared/functions/mpam/genMPAM

```
// genMPAM
// =======
// Returns MPAMinfo for exception level el.
// If InD is TRUE returns MPAM information using PARTID_I and PMG_I fields
// of MPAMel_ELx register and otherwise using PARTID_D and PMG_D fields.
// Produces a Secure PARTID if Secure is TRUE and a Non-secure PARTID otherwise.

MPAMinfo genMPAM(integer el, boolean InD, boolean secure)
    MPAMinfo returnInfo;
    PARTIDtype partidel;
    boolean perr;
    boolean gstplk = (el == 0 && EL2Enabled() &&
                      MPAMHCR_EL2.GSTAPP_PLK == '1' && HCR_EL2.TGE == '0');
```

```
    integer eff_el = if gstplk then 1 else el;
    (partidel, perr) = genPARTID(eff_el, InD);
    PMGtype groupel  = genPMG(eff_el, InD, perr);
    returnInfo.mpam_ns = if secure then '0' else '1';
    returnInfo.partid  = partidel;
    returnInfo.pmg     = groupel;
    return returnInfo;
```

shared/functions/mpam/genMPAMel

```
// genMPAMel
// =========
// Returns MPAMinfo for specified EL in the current security state.
// InD is TRUE for instruction access and FALSE otherwise.

MPAMinfo genMPAMel(bits(2) el, boolean InD)
    boolean secure = IsSecure();
    if HaveMPAMExt() && MPAMisEnabled() then
        return genMPAM(UInt(el), InD, secure);
    return DefaultMPAMinfo(secure);
```

shared/functions/mpam/genPARTID

```
// genPARTID
// =========
// Returns physical PARTID and error boolean for exception level el.
// If InD is TRUE then PARTID is from MPAMel_ELx.PARTID_I and
// otherwise from MPAMel_ELx.PARTID_D.

(PARTIDtype, boolean) genPARTID(integer el, boolean InD)
    PARTIDtype partidel = getMPAM_PARTID(el, InD);

    integer partid_max = UInt(MPAMIDR_EL1.PARTID_MAX);
    if UInt(partidel) > partid_max then
        return (DefaultPARTID, TRUE);

    if MPAMisVirtual(el) then
        return MAP_vPARTID(partidel);
    else
        return (partidel, FALSE);
```

shared/functions/mpam/genPMG

```
// genPMG
// ======
// Returns PMG for exception level el and I- or D-side (InD).
// If PARTID generation (genPARTID) encountered an error, genPMG() should be
// called with partid_err as TRUE.

PMGtype genPMG(integer el, boolean InD, boolean partid_err)
    integer pmg_max = UInt(MPAMIDR_EL1.PMG_MAX);

    // It is CONSTRAINED UNPREDICTABLE whether partid_err forces PMG to
    // use the default or if it uses the PMG from getMPAM_PMG.
    if partid_err then
        return DefaultPMG;
    PMGtype groupel = getMPAM_PMG(el, InD);
    if UInt(groupel) <= pmg_max then
        return groupel;
    return DefaultPMG;
```

shared/functions/mpam/getMPAM_PARTID

```
// getMPAM_PARTID
// ==============
```

```
// Returns a PARTID from one of the MPAMn_ELx registers.
// MPAMn selects the MPAMn_ELx register used.
// If InD is TRUE, selects the PARTID_I field of that
// register.  Otherwise, selects the PARTID_D field.

PARTIDtype getMPAM_PARTID(integer MPAMn, boolean InD)
    PARTIDtype partid;
    boolean el2avail = EL2Enabled();

    if InD then
        case MPAMn of
            when 3 partid = MPAM3_EL3.PARTID_I;
            when 2 partid = if el2avail then MPAM2_EL2.PARTID_I else Zeros();
            when 1 partid = MPAM1_EL1.PARTID_I;
            when 0 partid = MPAM0_EL1.PARTID_I;
            otherwise partid = PARTIDtype UNKNOWN;
    else
        case MPAMn of
            when 3 partid = MPAM3_EL3.PARTID_D;
            when 2 partid = if el2avail then MPAM2_EL2.PARTID_D else Zeros();
            when 1 partid = MPAM1_EL1.PARTID_D;
            when 0 partid = MPAM0_EL1.PARTID_D;
            otherwise partid = PARTIDtype UNKNOWN;
    return partid;
```

shared/functions/mpam/getMPAM_PMG

```
// getMPAM_PMG
// ===========
// Returns a PMG from one of the MPAMn_ELx registers.
// MPAMn selects the MPAMn_ELx register used.
// If InD is TRUE, selects the PMG_I field of that
// register.  Otherwise, selects the PMG_D field.

PMGtype getMPAM_PMG(integer MPAMn, boolean InD)
    PMGtype pmg;
    boolean el2avail = EL2Enabled();

    if InD then
        case MPAMn of
            when 3 pmg = MPAM3_EL3.PMG_I;
            when 2 pmg = if el2avail then MPAM2_EL2.PMG_I else Zeros();
            when 1 pmg = MPAM1_EL1.PMG_I;
            when 0 pmg = MPAM0_EL1.PMG_I;
            otherwise pmg = PMGtype UNKNOWN;
    else
        case MPAMn of
            when 3 pmg = MPAM3_EL3.PMG_D;
            when 2 pmg = if el2avail then MPAM2_EL2.PMG_D else Zeros();
            when 1 pmg = MPAM1_EL1.PMG_D;
            when 0 pmg = MPAM0_EL1.PMG_D;
            otherwise pmg = PMGtype UNKNOWN;
    return pmg;
```

shared/functions/mpam/mapvpmw

```
// mapvpmw
// =======
// Map a virtual PARTID into a physical PARTID using
// the MPAMVPMn_EL2 registers.
// vpartid is now assumed in-range and valid (checked by caller)
// returns physical PARTID from mapping entry.

PARTIDtype mapvpmw(integer vpartid)
    bits(64) vpmw;
    integer wd = vpartid DIV 4;
```

```
case wd of
    when 0 vpmw = MPAMVPM0_EL2;
    when 1 vpmw = MPAMVPM1_EL2;
    when 2 vpmw = MPAMVPM2_EL2;
    when 3 vpmw = MPAMVPM3_EL2;
    when 4 vpmw = MPAMVPM4_EL2;
    when 5 vpmw = MPAMVPM5_EL2;
    when 6 vpmw = MPAMVPM6_EL2;
    when 7 vpmw = MPAMVPM7_EL2;
    otherwise vpmw = Zeros(64);
// vpme_lsb selects LSB of field within register
integer vpme_lsb = (vpartid REM 4) * 16;
return vpmw<vpme_lsb +: 16>;
```

Appendix A
Generic Resource Controls

This chapter contains the following sections:

A.1 Introduction

This appendix is informative.

Several of the resource controls defined in this specification fit one of the generic models for resource controls in this appendix.

A.2 Portion resource controls

Some resources may be divided into fixed quanta, termed *portions*, that can be allocated for the exclusive use of a partition or shared between two or more partitions. Figure A-1 shows how partitions can have private and shared Portion Bit Map (PBM) allocations.

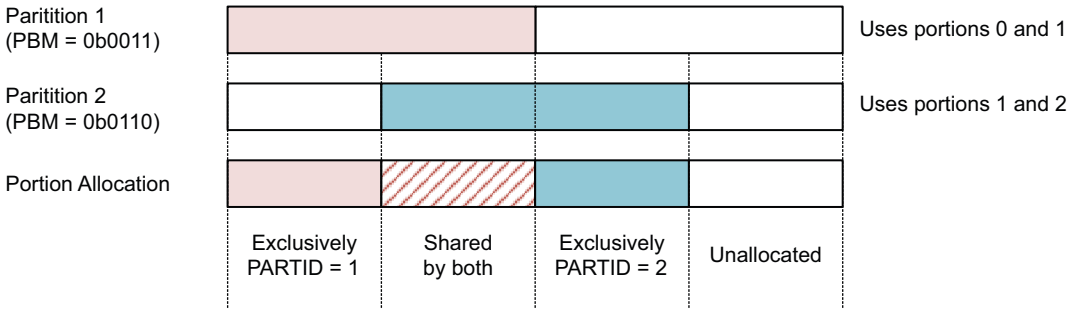


Figure A-1 Generic portion shared and exclusive allocations.

In portion resource controls, the control setting is a bitmap in which each bit corresponds to a particular portion of the resource, as shown in Figure A-2. Each bit grants the PARTID using this control setting to allocate the portion corresponding to that bit.

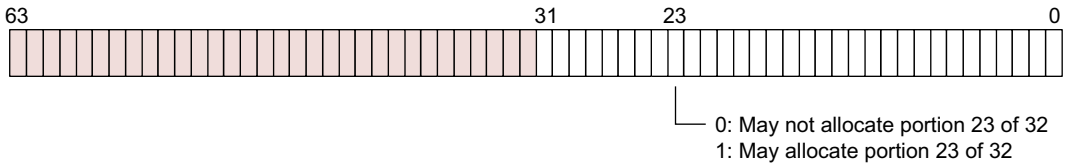


Figure A-2 Generic portion bit map.

PBMs may be wide. Generic PBMs could be up to 2^{15} bits in width.

A PBM is a vector of single-bit elements. Element 0 is bit 0 at the address (MPAMF_BASE + PBM_offset) where PBM_offset is the offset of the particular PBM register. Both the bitmap and the register to access the bitmap extend in length at increasing 32-bit word addresses for the width in bits of the PBM (PBM_WD). If the 32-bit word containing the highest byte of the bitmap (MPAMF_BASE + PBM_offset + (PBM_WD>>3)) has unused bits, those bits are RES0.

To access the PBM for portion n, access the 32-bit word of the PBM register at the address MPAMF_BASE + PBM_offset + ((n >> 3) & ~3). Then access bit (n & 31).

A.3 Maximum-usage resource controls

Many resources can be controlled by a maximum-usage resource control. With this control, resources may be allocated to a partition as long as the partition's maximum usage is not exceeded. If the maximum usage is reached, further allocation must be prevented, or deferred, or lowered in priority, or caused to reclaim a previous allocation, or caused to replace a previous allocation.

Maximum-usage control settings are a maximum fraction of the resource that the PARTID may use. The parameter is represented as a 16-bit fixed-point fraction of the capacity of the resource with a discoverable number of fractional bits. For example, if a resource has an 8-bit fractional width, bits [15:8] of the setting are used to control the resource allocation. To ensure that the range includes 100% of the resource, the control value is increased by 1 in the least significant implemented bit before being used to limit the usage to the maximum. See *About the fixed-point fractional format* on page 9-142 for the fixed-point fractional format.

A.4 Proportional resource allocation facilities

MPAM proportional stride partitioning is related to two software resource-management interfaces:

- The Linux cgroup weights interface assigns integer weights to indicate the relative proportion of the resource given to a process.
- The VMware shares interface similarly assigns an integer share to indicate the relative proportion of the resource that a virtual machine is given.

Weight and share values are positive integers. For example, Linux group weights are in the range of 1 to 10000, with a default value of 100.

The value of weight or share is used to compute the fraction of the resource, *f*, for partition, *p*, as:

$$f(p) = \frac{Weight_p}{\sum_{\text{all w}} Weight_w}$$

A partition's stride is the scaled reciprocal of its weight:

$$Stride \text{ of p} = \frac{S}{f(p)}$$

The scaling factor, S, should be chosen as equal to the largest f(p) so as to normalize stride values and give the smallest stride in the system = 1. All strides should be scaled by the same S.

Stride-based proportional allocation is well-suited to temporal or rate-of-occurrence resources, such as bandwidth.

The standard interface for proportional allocation is a positive unsigned integer, STRIDEM1, with an IMPLEMENTATION DEFINED field width of w. STRIDEM1 has the range [0 … 2^w-1] so stride has the range [1 … 2^w]. If a stride after normalization is greater than 2^w, it should be programmed into the control as 2^w – 1, the largest representable STRIDEM1.

Properties of proportional allocation include:

- Proportion of resource shrinks and grows as partitions come and go.
- Subdividable: If VM A has ½ fraction of the whole resource and its child application, y, has 2/3 fraction of the VM's resource, then y is given 1/2 * 2/3 == 1/3 fraction of the whole resource.
- Proportional allocation only needs to consider the current contenders for a temporal resource, such as memory bandwidth.
- A proportional allocation scheme is called *work-conserving* if it does not idle the resource when only low-proportion requests are available, but instead uses as much of the resource as it has requests to use. A proportional allocation scheme might allocate the resource to those lower-proportion requests, in proportion to their relative weights.

A.4.1 Model of stride-based memory bandwidth scheduling

This model is intended to explain the operation of stride-based memory bandwidth scheduling without dictating an implementation. Arm believes that a variety of implementations are possible.

In this model, each partition has an *offset[p]* that tracks the time since the partition, p, consumed bandwidth but is bounded to be less than *offset_limit*. When a request, *r*, arrives it is given a *deadline*, of the *current_time* plus *stride(p)* minus *offset(p)*. The *offset(p)* is set to current_time – deadline, and the *offset(p)* is incremented in event-time units until it reaches the *offset_limit*.

In the model, requests are serviced as quickly as possible in deadline order. Newly arriving requests with small strides (highest access to bandwidth) may go ahead of earlier requests with large strides.

If there are requests to process, this model does not prevent servicing a request with a distant future deadline if there are no requests available with earlier deadlines. As such, this model scheme is work-conserving.

A.5 Combining resource controls

Maximum-usage resource controls, portion resource controls, and other resource controls may coexist on the same resource. Combined resource controls should produce a combined effect. For example, combining portion control and maximum-usage control for the same resource should allocate the resource while satisfying both controls.

All resource controls should have at least one setting that does not limit access to the resource. When an implementation contains multiple controls for the same resource, the limits imposed on a partition's usage by each control are all applied. By selecting which controls limit a partition's usage and which do not, software can exercise a variety of regulation styles within a single system.

Appendix B
MSC Firmware Data

This chapter contains the following sections:

B.1 Introduction

In a system containing MPAM, discovery of the memory-system topology and certain implementation parameters of MPAM controls and monitors must be provided to MPAM-aware software via firmware data. The software-to-firmware interface to the MPAM firmware data is beyond the scope of this description. Examples of firmware data interfaces include:

- ACPI 2.0.
- Device Tree.

Firmware data for static devices can be pre-configured for an implementation and stored as part of the firmware, or it can be dynamically discovered through probing and other tests, or some combination of these two approaches.

B.2 Partitioning-control parameters

Table B-1 Partitioning-control parameters.

Control	Parameter	Data Format	Description
MPAM	MPAMF_BASE_NS	Address	Every MPAM-capable device has the MPAMF_IDR MMR at offset 0 from the MPAMF_BASE_NS in the Non-secure address space. Other MPAM memory-mapped registers are at known offsets from this address. See Chapter 11 *Memory-Mapped Registers*.
MPAM	MPAMF_BASE_S	Address	Every MPAM-capable device has the MPAMF_IDR MMR at offset 0 from the MPAMF_BASE__S in the Secure address space. Other MPAM memory-mapped registers are at known offsets from this address. See Chapter 11 *Memory-Mapped Registers*.

B.3 Performance-monitoring parameters

Table B-2 Performance-monitoring parameters

Monitor	Parameter	Data Format	Description
CSU	MAX_NRDY_USEC	Uint32	Maximum number of microseconds that the NRDY signal can remain 1 in the absence of additional reconfiguration of the monitor or writes to the MSMON_CSU register. This firmware value is the maximum time when NRDY can be 1, so that software can know this value. MSMON_CSU.VALUE is accurate and MSMON_CSU.NRDY is zero before MAX_NRDY_USEC microseconds have elapsed since the monitor was configured, reconfigured, or written.

Glossary

This glossary describes some of the terms that are used in this document. Some of these terms are unique to MPAM and are introduced in this document while others are standard terms that can be found in the Glossary of the Arm Architecture Reference Manual.

Abort
An exception caused by an illegal memory access. Aborts can be caused by the external memory system or the MMU.

Aligned
A data item stored at an address that is exactly divisible by the highest power of 2 that divides exactly into its size in bytes. Aligned halfwords, words and doublewords therefore have addresses that are divisible by 2, 4, and 8, respectively.

AMBA
Advanced Microcontroller Bus Architecture. The AMBA family of protocol specifications is the Arm open standard for on-chip buses. AMBA provides solutions for the interconnection and management of the functional blocks that make up a *System-on-Chip* (SoC). Applications include the development of embedded systems with one or more processors or signal processors and multiple peripherals.

Banked register
A register that has multiple instances, with the instance that is in use depending on the PE mode, Security state, or other PE state.

Burst
A group of transfers that form a single transaction. With AMBA protocols, only the first transfer of the burst includes address information, and the transfer type determines the addresses used for subsequent transfers.

BWA
BandWidth Allocation.

BWPBM
BandWidth Portion Bit Map.

CONSTRAINED UNPREDICTABLE
Where an instruction can result in UNPREDICTABLE behavior, the Armv8 architecture specifies a narrow range of permitted behaviors. This range is the range of CONSTRAINED UNPREDICTABLE behavior. All implementations that are compliant with the architecture must follow the CONSTRAINED UNPREDICTABLE behavior.

Execution at Non-secure EL1 or EL0 of an instruction that is CONSTRAINED UNPREDICTABLE can be implemented as generating a trap exception that is taken to EL2, provided that at least one instruction that is not UNPREDICTABLE and is not CONSTRAINED UNPREDICTABLE causes a trap exception that is taken to EL2.

In body text, the term CONSTRAINED UNPREDICTABLE is shown in SMALL CAPITALS.

See also UNPREDICTABLE.

Core See *Processing element (PE)*.

CPBM Cache-Portion Bit Map.

CSU Cache-Storage Usage.

DSB Data Synchronization Barrier.

E2H EL2 Host. A bit field in the HCR_EL2 register. This configuration executes a type-2 hypervisor and its host operating system in EL2 rather than EL1, for better performance.

Type-2 hypervisors run on a host operating system rather then running as a small, standalone OS-like program. For example, kvm is a type-2 hypervisor.

HCR An abbreviated reference to the Hypervisor Configuration Registers in AArch64 HCR_EL2 and in AArch32 HCR and HCR2.

ICN InterConnect Network.

ID An identifier or label.

Intermediate physical address (IPA)
An implementation of virtualization, the address to which a Guest OS maps a VA. A hypervisor might then map the IPA to a PA. Typically, the Guest OS is unaware of the translation from IPA to PA.

See also Physical address (PA), Virtual address (VA).

IPA See *Intermediate physical address (IPA)*.

kvm Kernel-based Virtual Machine, an open-source software package that implements a type-2 hypervisor within Linux.

LPI Locality-specific Peripheral Interrupt.

MBWU Memory BandWidth Usage.

Memory-system component
MSC. A function, unit, or design block in a memory system that can have partitionable resources. MSCs consist of all units that handle load or store requests issued by any MPAM master. These include cache memories, interconnects, memory management units, memory channel controllers, queues, buffers, rate adaptors, etc. An MSC may contain one or more resources that each may have zero or more resource partitioning controls. For example, a PE may contain several caches, each of which might have zero or more resource partitioning controls.

Memory-system resource
A resources that affects the performance of software's use of the memory system and is either local to an MSC (such as cache-memory capacity) or non-local (such as memory bandwidth, which is present over an entire path, from master to slave, that may pass through multiple MSCs).

MMR Memory-Mapped Register.

MPAM Memory System Resource Partitioning and Monitoring.

MPAM information
The MPAM information bundle, comprising PARTID, PMG, and MPAM_NS.

MPAM_NS MPAM security-space bit. It is not stored in a PE register; it comes from the current security state of a PE and is communicated to MSCs as part of the MPAM information bundle. In non-PE masters, the security state can be determined in other ways.

MSC Memory-System Component. See above.

NRDY Not-Ready bit. MPAM resource monitors set this bit to indicate that the monitor register does not currently have an accurate value.

NS Non-Secure. A bit indicating that an address space is not Secure.

PA See *Physical address (PA)*.

PARTID Partition ID. Together with the MPAM_NS bit, it selects a memory-system resource partition to use in the MSCs. For each resource with a resource partitioning control in each MSC, the PARTID and MPAM_NS select resource control levels, limits, or allocations from local control-setting tables.

Partition A division of resources. A partition is manifest in a PARTID and MPAM_NS. In an MSC, the PARTID and MPAM_NS select partitioning control settings that affect the partitioning by regulating the allocation of the resource to requests using that PARTID and MPAM_NS.

PE *See Processing element (PE)*.

Physical address (PA)
An address that identifies a location in the physical memory map.

See also Intermediate physical address (IPA), Virtual address (VA).

Physical PARTID
A partition ID that is transmitted with memory requests and can be used by MSCs to control resources usage. A physical PARTID is in either the Non-secure or Secure PARTID space.

PMG Performance Monitoring Group, a property of a partition used in MSCs by MPAM performance monitors that can be programmed to be sensitive to the particular PARTID and PMG combination.

Portion A uniquely identifiable part of the resource. It is of fixed size or capacity. A particular resource has a constant number of portions. Portions are distinct. Portion n is the same part of the resource for every partition. Thus, every partition that is given access to a portion n shares access to portion n.

PPI Private Peripheral Interrupt.

Processing element (PE)
The abstract machine defined in the Arm architecture, as documented in an Arm Architecture Reference Manual. A PE implementation compliant with the Arm architecture must conform with the behaviors described in the corresponding Arm Architecture Reference Manual.

RAZ *See Read-As-Zero (RAZ)*.

RAZ/WI Read-As-Zero, Writes Ignored.

Hardware must implement the field as Read-as-Zero, and must ignore writes to the field.

Software can rely on the field reading as all 0s, and on writes being ignored.

This description can apply to a single bit that reads as 0, or to a field that reads as all 0s.

See also Read-As-Zero (RAZ).

Read-As-Zero (RAZ)
Hardware must implement the field as reading as all 0s.

Software:

- Can rely on the field reading as all 0s
- Must use a SBZP policy to write to the field.

This description can apply to a single bit that reads as 0, or to a field that reads as all 0s.

See also RAZ/WI, RES0.

RES0 A reserved bit. Used for fields in register descriptions, and for fields in architecturally-defined data structures that are held in memory, for example in translation table descriptors.

Within the architecture, there are some cases where a register bit or field:

- Is RES0 in some defined architectural context.

- Has different defined behavior in a different architectural context.

Note

- RES0 is not used in descriptions of instruction encodings.
- Where an AArch32 System register is Architecturally mapped to an AArch64 System register, and a bit or field in that register is RES0 in one Execution state and has defined behavior in the other Execution state, this is an example of a bit or field with behavior that depends on the architectural context.

This means the definition of RES0 for fields in read/write registers is:

If a bit is RES0 in all contexts

For a bit in a read/write register, it is IMPLEMENTATION DEFINED whether:

1. The bit is hardwired to 0. In this case:
   - Reads of the bit always return 0.
   - Writes to the bit are ignored.
2. The bit can be written. In this case:
   - An indirect write to the register sets the bit to 0.
   - A read of the bit returns the last value successfully written, by either a direct or an indirect write, to the bit.

     If the bit has not been successfully written since reset, then the read of the bit returns the reset value if there is one, or otherwise returns an UNKNOWN value.
   - A direct write to the bit must update a storage location associated with the bit.
   - The value of the bit must have no effect on the operation of the PE, other than determining the value read back from the bit, unless this Manual explicitly defines additional properties for the bit.

Whether RES0 bits or fields follow behavior 1 or behavior 2 is IMPLEMENTATION DEFINED on a field-by-field basis.

If a bit is RES0 only in some contexts

For a bit in a read/write register, when the bit is described as RES0:

- An indirect write to the register sets the bit to 0.
- A read of the bit must return the value last successfully written to the bit, by either a direct or an indirect write, regardless of the use of the register when the bit was written.

  If the bit has not been successfully written since reset, then the read of the bit returns the reset value if there is one, or otherwise returns an UNKNOWN value.
- A direct write to the bit must update a storage location associated with the bit.
- While the use of the register is such that the bit is described as RES0, the value of the bit must have no effect on the operation of the PE, other than determining the value read back from that bit, unless this Manual explicitly defines additional properties for the bit.

Considering only contexts that apply to a particular implementation, if there is a context in which a bit is defined as RES0, another context in which the same bit is defined as RES1, and no context in which the bit is defined as a functional bit, then it is IMPLEMENTATION DEFINED whether:

- Writes to the bit are ignored, and reads of the bit return an UNKNOWN value.
- The value of the bit can be written, and a read returns the last value written to the bit.

The RES0 description can apply to bits or fields that are read-only, or are write-only:

- For a read-only bit, RES0 indicates that the bit reads as 0, but software must treat the bit as UNKNOWN.
- For a write-only bit, RES0 indicates that software must treat the bit as SBZ.

A bit that is RES0 in a context is reserved for possible future use in that context. To preserve forward compatibility, software:

- Must not rely on the bit reading as 0.

- Must use an SBZP policy to write to the bit.

This RES0 description can apply to a single bit, or to a field for which each bit of the field must be treated as RES0.

In body text, the term RES0 is shown in SMALL CAPITALS.

See also Read-As-Zero (RAZ), RES1, UNKNOWN.

RES1 A reserved bit. Used for fields in register descriptions, and for fields in architecturally-defined data structures that are held in memory, for example in translation table descriptors.

Within the architecture, there are some cases where a register bit or field:

- Is RES1 in some defined architectural context.
- Has different defined behavior in a different architectural context.

Note

- RES1 is not used in descriptions of instruction encodings.
- Where an AArch32 System register is Architecturally mapped to an AArch64 System register, and a bit or field in that register is RES1 in one Execution state and has defined behavior in the other Execution state, this is an example of a bit or field with behavior that depends on the architectural context.

This means the definition of RES1 for fields in read/write registers is:

If a bit is RES1 in all contexts

For a bit in a read/write register, it is IMPLEMENTATION DEFINED whether:

1. The bit is hardwired to 1. In this case:
   - Reads of the bit always return 1.
   - Writes to the bit are ignored.
2. The bit can be written. In this case:
   - An indirect write to the register sets the bit to 1.
   - A read of the bit returns the last value successfully written, by either a direct or an indirect write, to the bit.

     If the bit has not been successfully written since reset, then the read of the bit returns the reset value if there is one, or otherwise returns an UNKNOWN value.
   - A direct write to the bit must update a storage location associated with the bit.
   - The value of the bit must have no effect on the operation of the PE, other than determining the value read back from the bit, unless this Manual explicitly defines additional properties for the bit.

Whether RES1 bits or fields follow behavior 1 or behavior 2 is IMPLEMENTATION DEFINED on a field-by-field basis.

If a bit is RES1 only in some contexts

For a bit in a read/write register, when the bit is described as RES1:

- An indirect write to the register sets the bit to 1.
- A read of the bit must return the value last successfully written to the bit, regardless of the use of the register when the bit was written.

  Note

  As indicated in this list, this value might be written by an indirect write to the register.

  If the bit has not been successfully written since reset, then the read of the bit returns the reset value if there is one, or otherwise returns an UNKNOWN value.
- A direct write to the bit must update a storage location associated with the bit.

- While the use of the register is such that the bit is described as RES1, the value of the bit must have no effect on the operation of the PE, other than determining the value read back from that bit, unless this Manual explicitly defines additional properties for the bit.

Considering only contexts that apply to a particular implementation, if there is a context in which a bit is defined as RES0, another context in which the same bit is defined as RES1, and no context in which the bit is defined as a functional bit, then it is IMPLEMENTATION DEFINED whether:

- Writes to the bit are ignored, and reads of the bit return an UNKNOWN value.
- The value of the bit can be written, and a read returns the last value written to the bit.

The RES1 description can apply to bits or fields that are read-only, or are write-only:

- For a read-only bit, RES1 indicates that the bit reads as 1, but software must treat the bit as UNKNOWN.
- For a write-only bit, RES1 indicates that software must treat the bit as SBO.

A bit that is RES1 in a context is reserved for possible future use in that context. To preserve forward compatibility, software:

- Must not rely on the bit reading as 1.
- Must use an SBOP policy to write to the bit.

This RES1 description can apply to a single bit, or to a field for which each bit of the field must be treated as RES1.

In body text, the term RES1 is shown in SMALL CAPITALS.

See also RES0, UNKNOWN.

Reserved

Unless otherwise stated:

- Instructions that are reserved or that access reserved registers have UNPREDICTABLE or CONSTRAINED UNPREDICTABLE behavior.
- Bit positions described as reserved are:
  - In an RW or WO register, RES0.
  - In an RO register, UNK.

SCR

Part of the name of a Secure Configuration Register.

SMMU

System Memory-Management Unit.

SPE

Statistical Profiling Extension.

SPI

Shared Peripheral Interrupt.

TGE

Trap General Exception. A field in the HCR_EL2 register. It causes EL0 exceptions, that would normally trap to EL1, to instead trap to EL2. This function can be used to run an EL2 host's applications at EL0, so that any exceptions in the application trap to the host OS at EL2.

UNDEFINED

Indicates cases where an attempt to execute a particular encoding bit pattern generates an exception, that is taken to the current Exception level, or to the default Exception level for taking exceptions if the UNDEFINED encoding was executed at EL0. This applies to:

- Any encoding that is not allocated to any instruction.
- Any encoding that is defined as never accessible at the current Exception level.
- Some cases where an enable, disable, or trap control means an encoding is not accessible at the current Exception level.

If the generated exception is taken to an Exception level that is using AArch32 then it is taken as an Undefined Instruction exception.

Note

On reset, the default Exception level for taking exceptions from EL0 is EL1. However, an implementation might include controls that can change this, effectively making EL1 inactive. See the description of the Exception model for more information

In body text, the term UNDEFINED is shown in SMALL CAPITALS.

UNKNOWN

An UNKNOWN value does not contain valid data, and can vary from moment to moment, instruction to instruction, and implementation to implementation. An UNKNOWN value must not return information that cannot be accessed at the current or a lower level of privilege using instructions that are not UNPREDICTABLE, are not CONSTRAINED UNPREDICTABLE, and do not return UNKNOWN values.

An UNKNOWN value must not be documented or promoted as having a defined value or effect.

In body text, the term UNKNOWN is shown in SMALL CAPITALS.

See also CONSTRAINED UNPREDICTABLE, UNDEFINED, , UNPREDICTABLE.

UNPREDICTABLE

Means the behavior cannot be relied upon. UNPREDICTABLE behavior must not perform any function that cannot be performed at the current or a lower level of privilege using instructions that are not UNPREDICTABLE.

UNPREDICTABLE behavior must not be documented or promoted as having a defined effect.

An instruction that is UNPREDICTABLE can be implemented as UNDEFINED.

Execution at Non-secure EL1 or EL0 of an instruction that is UNPREDICTABLE can be implemented as generating a trap exception that is taken to EL2, provided that at least one instruction that is not UNPREDICTABLE and is not CONSTRAINED UNPREDICTABLE causes a trap exception that is taken to EL2.

In body text, the term UNPREDICTABLE is shown in SMALL CAPITALS.

See also CONSTRAINED UNPREDICTABLE, UNDEFINED.

VA

See Virtual address (VA).

Virtual address (VA)

An address generated by an Arm PE. This means it is an address that might be held in the program counter of the PE. For a PMSA implementation, the virtual address is identical to the physical address.

See also Intermediate physical address (IPA), Physical address (PA).

Virtual PARTID

One of a small range of PARTIDs that can be used by a virtual machine (VM). Virtual PARTIDs are mapped into physical PARTIDs using the virtual partition mapping entries in the MPAMVPM0 - MPAMVPM7 registers.

VM

Virtual Machine.

VMM

Virtual Machine Monitor. An alias for “hypervisor”.

Word

A 32-bit data item. Words are normally word-aligned in Arm systems.

Word-aligned

Means that the address is divisible by 4.