Arm® Architecture Reference Manual Supplement

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

ARM DDI 0598C.b (ID012521)

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

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
21 February 2020	B.a	Non-Confidential	Updated EAC release incorporating MPAMv0.1 and MPAMv1.1 architectures
17 July 2020	B.b	Non-Confidential	Updated EAC release
22 January 2021	B.c	Non-Confidential	Updated EAC release
23 June 2021	C.a	Non-Confidential	Updated EAC release incorporating MPAM for Realm Management Extension
19 October 2021	C.b	Non-Confidential	Updated EAC release integrating MPAM for Realm Management Extension

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

(LES-PRE-20349 version 21.0)

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.

The information in this manual is at EAC quality, which means that all features of the specification are described in the manual.

Web Address

`http://www.arm.com`

Contents

Preface

This preface introduces the MPAM Extension architecture specification. It contains the following sections:

About this book

This book is the *Architecture Specification* for the *MPAM Extension Architecture Specification v1.0*, *v1.1,* and *v0.1*.

It specifies:

- System registers and behaviors for generation of MPAM information in processing elements, or PEs.
- Memory-mapped registers and standard types of resource control interfaces for Memory-System Components, or MSCs.
- Memory-mapped registers and resource usage monitors for measuring resource usage in MSCs.

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 all versions of the MPAM Extension. For more information on MPAM Extension versions, see *Versions of the MPAM Extension* on page 1-22.

This document primarily covers only the AArch64 Execution state, but the MPAM Extension does continue to operate in AArch32 state, as detailed in *MPAM AArch32 interoperability* on page 6-81.

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 optional features to implement in either a PE or an MSC.
- What resources in which MSCs should be controlled by MPAM.

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:

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 *Memory System 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 *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*.

———————————

How to read this book

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 8, 10, 11, and 12.

Readers interested in changes made by the Armv8.6 architecture extension should read sections mentioned in *Versions of the MPAM Extension* on page 1-22.

Readers interested in MPAM pseudocode should read Chapter 13.

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

Readers interested in Realm Management Extension, RME, should read the *Arm® Architecture Reference Manual Supplement, The Realm Management Extension (RME), for Armv9-A* (ARM DDI 0615).

Conventions

The following sections describe conventions that this book can use:

- *Typographic conventions* on page xiii.
- *Signals* on page xiii.
- *Numbers* on page xiii.
- *Pseudocode descriptions* on page xiii.

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://developer.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* on page xiii.
- 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 Chapter 13 *Pseudocode*.

Additional reading

This section lists relevant publications from Arm and third parties.

See Arm Developer, https://developer.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)
- *Arm® Architecture Reference Manual Supplement Armv9, for Armv9-A architecture profile* (ARM DDI 0608)
- *Arm® CoreSight Architecture Specification v2.0* (ARM IHI 0029D)
- *ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0* (ARM IHI 0069).
- *Arm® System Memory Management Unit Architecture Specification, SMMU architecture versions 3.0, 3.1 and 3.2* (ARM IHI 0070C.a).
- *Arm® Architecture Reference Manual Supplement, The Realm Management Extension (RME), for Armv9-A* (ARM DDI 0615).
- *Arm® System Memory Management Unit Architecture Specification, SMMU architecture* (ARM IHI 0070).
- *The Realm Management Extension (RME), for SMMUv3 Arm® System Memory Management Unit Architecture Supplement* (ARM IHI 0094).

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, *Arm® Architecture Reference Manual Supplement Memory System Resource Partitioning and Monitoring (MPAM), for A-profile architecture.*
- The number, ARM DDI 0598C.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.

Progressive Terminology Commitment

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

Previous issues of this document included terms that can be offensive. We have replaced these terms. If you find offensive terms in this document, please contact `terms@arm.com`.

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.

There are different versions of this MPAM Extension. For more information, see *Versions of the MPAM Extension* on page 1-22.

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 in future MPAM Extension specifications and 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 monitoring 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 Requester. 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 Versions of the MPAM Extension

This document describes several version of the MPAM architecture. The identification of architecture versions and the features present within a version differ between PEs and MSCs are described in:

- *MPAM versions for PEs* on page 1-22.
- *MPAM versions for MSCs* on page 1-22.
- *Relationships between MPAM versions* on page 1-27.
- *Interoperation of components with different MPAM versions* on page 1-28.

1.5.1 MPAM versions for PEs

There are multiple different versions of the MPAM Extension. The architecture version of the MPAM Extension implemented in a PE is given in ID_AA64PFR0_EL1.MPAM for the major version and ID_AA64PFR1_EL1.MPAM_frac for the minor version. Table 1-1 on page 1-22 shows how ID_AA64PFR0_EL1.MPAM and ID_AA64PFR1_EL1.MPAM_frac values indicate the MPAM architecture version.

Table 1-1 MPAM Extension implemented by a PE

ID_AA64PFR0_EL1.MPAM	ID_AA64PFR1_EL1.MPAM_frac	MPAM Extension Architecture version	Notes
`0b0000`	`0b0000`	None	MPAM is not implemented.
`0b0000`	`0b0001`	v0.1	MPAM v0.1 is implemented. MPAM v0.1 is the same as MPAM v1.1 with FORCE_NS which is incompatible with MPAM v1.0.
`0b0001`	`0b0000`	v1.0	MPAM v1.0 is implemented.
`0b0001`	`0b0001`	v1.1	MPAM v1.1 is implemented. MPAM v1.1 includes all features of MPAM v1.0. It must not include FORCE_NS.

The optional MPAM features and MPAM identifier sizes supported by a PE that supports a version of the MPAM Extension are indicated in the fields of MPAMIDR_EL1.

1.5.2 MPAM versions for MSCs

The architecture version of the MPAM Extension implemented in an MSC is given in the MPAMF_AIDR register fields, ArchMajorRev and ArchMinorRev. The MPAM Extension versions used in MSCs are a subset of the versions used in PEs because the MPAM MSC architecture does not cover the generation of MPAM information by MSCs that are not PEs. The architecture of the component specifies how that component generates MPAM information for memory system requests that it originates.

MPAM Extension versions and the corresponding values of fields in MPAMF_AIDR of the MSC are shown in Table 1-2 on page 1-23:

Table 1-2 MPAM version implemented by an MSC

MPAMF_AIDR		MPAM Extension version supported	MSC MPAM support
ArchMajor Rev	ArchMinor Rev		
`0b0000`	`0b0000`	None	The MSC does not implement MPAM.
`0b0000`	`0b0001`	N/A	Not a valid MPAM version for an MSC.
`0b0001`	`0b0000`	v1.0	The MSC implements MPAM v1.0 with features as described in the 32-bit MPAMF_IDR.
`0b0001`	`0b0001`	v1.1	The MSC implements MPAM v1.1 with features as described in the 64-bit MPAMF_IDR. MPAM v1.1 includes all of the MSC MPAM features of MPAM v1.0 plus additional MPAM features.

Most MPAM features in an MSC are optional. The particular MPAM features available in an MSC are described in the MSC's MPAMF_IDR register.

MPAMF_IDR is 32-bits in MPAM v1.0 and is 64-bits in MPAM v1.1.

MPAMF_IDR is permitted to have different MPAM features in different address spaces. If the MPAM feature RIS is implemented MPAMF_IDR is also permitted to have different features for different Resource Instances in an MSC.

MSCs can be used in MPAM v1.0 and v1.1, and in v0.1 under certain conditions. For more information on the conditions on use of MSCs in MPAM v0.1, see *MPAM versions in MSCs* on page 8-156.

If an MSC does not implement any of the MPAM v1.1 MSC features listed in *MPAM versions for MSCs* on page 1-22, then the MSC is of MPAM v1.0.

MSC of MPAM v1.1

The MPAM features that can be implemented in an MSC of MPAM v1.1 are:

Expansion of MPAMF_IDR

MPAMF_IDR is expanded to 64 bits to support bits that indicate the presence of features added from MPAM v1.1.

This feature is mandatory when the MSC implements MPAM v1.1.

This feature is implemented when MPAMF_IDR.EXT is set to 1.

For more information, see *MPAMF_IDR, MPAM Features Identification Register* on page 11-221.

Capturing of IMPLEMENTATION DEFINED resource partitioning controls or resource monitoring

This feature defines two fields that allow discovery of any IMPLEMENTATION DEFINED resource partitioning controls or IMPLEMENTATION DEFINED resource monitors that are implemented.

This feature is mandatory when the MSC implements MPAM v1.1 and MPAMF_IDR.HAS_IMPL_IDR is 1.

This feature is implemented when MPAMF_IDR.EXT is 1. Furthermore:

- When MPAMF_IDR.NO_IMPL_PART is 1, MPAMF_IMPL_IDR does not include the description of any implementation-specific resource partitioning controls.

- When MPAMF_IDR.NO_IMPL_MSMON is 1, MPAMF_IMPL_IDR does not include the description of any implementation-specific resource monitors.

For more information, see *MPAMF_IDR, MPAM Features Identification Register* on page 11-221.

Resource instance selection

Resource instance selection, or RIS, provides access to the control settings of multiple resources of the same type within one MSC.

This feature is optional when the MSC implements MPAM v1.1.

This feature is implemented when MPAMF_IDR.EXT and MPAMF_IDR.HAS_RIS are 1.

For more information, see

- *Resource instance selection* on page 8-158.
- *MPAMCFG_PART_SEL, MPAM Partition Configuration Selection Register* on page 11-277.
- *MSMON_CFG_MON_SEL, MPAM Monitor Instance Selection Register* on page 11-303.
- *Error conditions in accessing memory-mapped registers* on page 12-365.

Greater range for MBWU monitors

This feature supports 44-bit and 63-bit memory bandwidth usage counters.

This feature is optional when the MSC implements MPAM v1.1.

This feature is implemented when MPAMF_MBWUMON_IDR.HAS_LONG is 1.

For more information, see *Long MBWU counter and capture* on page 10-194.

Discovery of MPAMF_ESR and MPAMF_ECR

This feature supports the MPAMF_IDR.HAS_ESR field. This field indicates whether MPAMF_ESR and MPAMF_ECR are implemented.

This feature is mandatory when the MSC implements MPAM v1.1.

This feature is implemented when MPAMF_IDR.EXT is 1.

For more information, see *MPAMF_IDR, MPAM Features Identification Register* on page 11-221.

Expansion of MPAMF_ESR

This feature widens MPAMF_ESR to 64 bits to include space for a RIS field.

This feature is optional when the MSC implements MPAM v1.1. Implementation of this feature is mandatory if MPAMF_IDR.{HAS_ESR, HAS_RIS} are 1.

This feature is implemented when MPAMF_IDR.{EXT, HAS_EXTD_ESR} are 1.

For more information, see

- *MPAMF_ESR, MPAM Error Status Register* on page 11-359
- *Resource instance selection* on page 8-158.

1.5.3 MPAM system features by MPAM version

MPAM System features are described in on page 2-31 on page 1-17 on page 2-31chapters 2 through 5 in this supplement, MPAM system features that vary by version are described in Table 1-3 on page 1-24:

Table 1-3 System features by MPAM version

MPAM feature	MPAM v0.1	MPAM v1.0	MPAM v1.1	MPAM for RME
MPAM_NS signal	Required	Required	Required	Prohibited
MPAM_SP signal	Prohibited	Prohibited	Prohibited	Required

1.5.4 MPAM PE features by MPAM version

The features applicable for different MPAM versions are described in chapters 6 and 7. These features are summarized in Table 1-4 on page 1-25:

Table 1-4 MPAM PE features by MPAM version

MPAM feature	MPAM v0.1	MPAM v1.0	MPAM v1.1	MPAM for RME	ID field
PARTID Virtualization	Optional	Optional	Optional	Optional	Prohibited
Force secure PARTID to NS	Required	Prohibited	Prohibited	Prohibited	MPAMIDR_EL1.HAS_FORCE_NS
Secure Default PARTID	Optional	Prohibited	Optional	Optional	MPAMIDR_EL1.HAS_SDEFLT
TIDR in MPAM2_EL2	Required	Prohibited	Required	Required	MPAMIDR_EL1.HAS_TIDR
Four PARTID Spaces	Prohibited	Prohibited	Prohibited	Required	MPAMIDR_EL1.SP4
Alternative PARTID spaces	Prohibited	Prohibited	Prohibited	Required	MPAMIDR_EL1.HAS_ALTSP

1.5.5 MSC features by MPAM version

MPAM MSC features are covered by chapters 8 through 12 of this supplement.

MPAM MSC features by MPAM version are shown in Table 1-5 on page 1-25:

Table 1-5 MSC features by MPAM version

MPAM feature	Subordinate feature	Subordinate feature 2	MPAM v1.0	MPAM v1.1	MPAM for RME	ID field
Cache capacity partitioning			Optional	Optional	Optional	MPAMF_IDR.HAS_CCAP_PART
Cache portion partitioning			Optional	Optional	Optional	MPAMF_IDR.HAS_CPOR_PART
Memory BW partitioning			Optional	Optional	Optional	MPAMF_IDR.HAS_MBW_PART
	Minimum BW partioning		Optional	Optional	Optional	MPAMF_MBW_IDR.HAS_MIN
	Maximum BW partioning		Optional	Optional	Optional	MPAMF_MBW_IDR.HAS_MAX
	BW portion partitioning		Optional	Optional	Optional	MPAMF_MBW_IDR.HAS_PBM
	Proportional BW partitioning		Optional	Optional	Optional	MPAMF_MBW_IDR.HAS_PROP
	BW window writable		Optional	Optional	Optional	MPAMF_MBW_IDR.WINDWR
Priority positioning			Optional	Optional	Optional	MPAMF_IDR.HAS_PRI_PART

Table 1-5 MSC features by MPAM version (continued)

MPAM feature	Subordinate feature	Subordinate feature 2	MPAM v1.0	MPAM v1.1	MPAM for RME	ID field
	Internal priority partitioning		Optional	Optional	Optional	MPAMF_PRI_IDR.HAS_INTPRI
	Downstream priority partitioning		Optional	Optional	Optional	MPAMF_PRI_IDR.HAS_DSPRI
Memory Sys resource monitoring			Optional	Optional	Optional	MPAMF_IDR.HAS_MSMON
	Cache storage usage monitoring		Optional	Optional	Optional	MPAMF_MSMON_IDR.MSMON_CSU
		CSU monitor capture	Optional	Optional	Optional	MPAMF_CSUMON_IDR.HAS_CAPTURE
		CSU monitor read-only	Optional	Optional	Optional	MPAMF_CSUMON_IDR._CSU_RO
	Memory BW usage monitoring		Optional	Optional	Optional	MPAMF_MSMON_IDR. MSBMON_MBWU
		MBWU monitor capture	Optional	Optional	Optional	MPAMF_MBWUMON_IDR.HAS_CAPTURE
		MBWU monitor Long	Optional	Optional	Optional	MPAMF_MBWUMON_IDR.HAS_LONG
		MBWU monitor R/W filtering	Optional	Optional	Optional	MPAMF_MBWUMON_IDR.HAS_RWBW
		MBWU monitor scaling	Optional	Optional	Optional	MPAMF_MBWUMON_IDR. SCALE
	Monitor overflow status register		Prohibited	Optional	Optional	MPAMF_MSMON_IDR.HS_OFLOW_SR
	Monitor overflow MSI		Prohibited	Optional	Optional	MPAMF_MSMON_IDR.OFLW_MSI
	No hardwired overflow interrupt		Prohibited	Optional	Optional	MPAMF_MSMON_IDR.NO_OFLOWINTR
	Local monitor capture event generator		Optional	Optional	Optional	MPAMF_MSMON_IDR.HAS_LOCAL_CAPT_EVNT
PARTID narrowing			Optional	Optional	Optional	MPAMF_IDR.HAS_PARTID_NRW
Implementation-defined ID Reg			Optional	Optional	Optional	MPAMF_IDR.HAS_IMPL_IDR
	Impl IDR no partitioning		Prohibited	Required	Required	MPAMF_IDR.IMPL_IDR
	Impl IDR no monitoring		Prohibited	Required	Required	MPAMF_IDR.IMPL_MSMON

Table 1-5 MSC features by MPAM version (continued)

MPAM feature	Subordinate feature	Subordinate feature 2	MPAM v1.0	MPAM v1.1	MPAM for RME	ID field
Extended ID register			Prohibited	Required	Required	MPAMF_IDR.EXT
Resource instance selector			Prohibited	Optional	Optional	MPAMF_IDR.HAS_RIS
Error status register			Prohibited	Optional	Optional	MPAMF_IDR.HAS_ESR
	Extended error status register		Prohibited	Optional	Optional	MPAMF_IDR.HAS_EXTD_ESR
Error MSI			Prohibited	Optional	Optional	MPAMF_IDR.HAS_ERR_MSI
Four PARTID spaces					Required	MPAMF_IDR.SP4

1.5.6 Relationships between MPAM versions

This section gives the relationships between MPAM versions.

MPAM v0.1

An MPAM v0.1 PE implements any permitted subset of the features of MPAM v1.1 and also implements MPAM3_EL3.FORCE_NS. The FORCE_NS field cannot be present in any other MPAM version.

In a PE that implements MPAM v0.1, the MPAM features available (either Required or Optional) are described in Table 1-4 on page 1-25.

In an MSC that implements MPAM v0.1, the MPAM features available are (either Required or Optional) are described in Table 1-5 on page 1-25.

For more information see *SDEFLT and FORCE_NS settings to control Secure MPAM PARTID use* on page 6-73 and *MPAM3_EL3, MPAM3 Register (EL3)* on page 7-110 for FORCE_NS.

MPAM v1.0

MPAM v1.0 is the base version of MPAM. Unless explicitly defined, all features from MPAM v1.0 are present in the other versions of MPAM.

In a PE that implements MPAM v1.0, the MPAM features available (either Required or Optional) are described in Table 1-4 on page 1-25.

In an MSC that implements MPAM v1.0, the MPAM features available (either Required or Optional) are described in Table 1-5 on page 1-25.

MPAM v1.1

MPAM v1.1 adds features beyond the base version of MPAM. Unless explicitly removed, all features from MPAM v1.1 are present in MPAM v0.1 and in MPAM for RME.

In a PE that implements MPAM v1.1, the MPAM features available (either Required or Optional) are described in Table 1-4 on page 1-25.

In an MSC that implements MPAM v1.1, the MPAM features available (either Required or Optional) are described in Table 1-5 on page 1-25.

MPAM for RME

The MPAM for RME architecture supports the Realm Management Extension (RME) in systems, PEs and MSCs.

MPAM for RME requires MPAM v1.1 or higher.

In a PE that implements both RME and MPAM, MPAM for RME is required.

In a PE, MPAM for RME requires the MPAM feature ALTSP.

In a PE that implements MPAM for RME, the MPAM features available (either Required or Optional) are described in Table 1-4 on page 1-25.

In an MSC that implements MPAM for RME, the MPAM features available (either Required or Optional) are described in Table 1-5 on page 1-25.

An MPAM for RME implementation requires support for 4 PARTID spaces, see *MPAM for RME propagation of MPAM_SP with requests* on page 4-46.

1.5.7 Interoperation of components with different MPAM versions

Hardware should not prevent PEs that implement different versions of the MPAM architecture to coexist within a system. However, PEs that implement different versions of the MPAM architecture might present a difficulty for software.

Hardware should not prevent MSCs that implement different versions of the MPAM architecture to coexist within a system.

There is no required relationship between the MPAM architecture version of a PE and the MPAM architecture version of an MSC accessed by that PE.

1.6 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 *Versions of the MPAM Extension* on page 1-22.

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.

1.7 Example uses

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

1.7.1 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.

1.7.2 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.

1.7.3 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 *informative*.

MPAM partitioning of memory-system performance resources must not affect the correctness of any memory behavior specified in the *Arm Architecture Reference Manual Armv8, for Armv8-A architecture profile*. The Armv8-A memory model, as specified in that manual, must be followed in all of its particulars, including requirements for observation, coherence, caching, order, atomicity, endianness, alignment, memory types, and any other requirements defined in the Armv8-A memory model. Furthermore, these requirements must also be met:

- For single-PE and multiple-PE environments.
- When the MPAM information in multiple requests to an MSC are the same or are different, and whether those multiple requests come from a single requestor or from 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 Armv8, for Armv8-A architecture profile*.

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 Requesters 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 or PARTID. The architectural maximum width of a PARTID field is 16 bits.

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

3.3 Physical address spaces and Security state

The Armv8-A architecture defines two physical address spaces:

- Non-secure physical address space.
- Secure physical address space.

RME provides two additional physical address spaces:

- Realm physical address space.
- Root physical address space.

MPAM makes use of the physical address spaces to access the resource control settings in memory system components. The controls for each PARTID space are accessed in the physical address space associated with that PARTID space.

Table 3-1 Physical address spaces to access MSC control settings for PARTID spaces

Physical address space	Access to control settings for PARTID space
Non-secure physical address space	Non-secure PARTID space
Secure physical address space	Secure PARTID space
Realm physical address space	Realm PARTID space
Root physical address space	Root PARTID space

3.4 PARTID spaces and properties

MPAM uses two PARTID spaces to label memory system requests:

- Secure physical PARTID space. This space is accessed when a Requester is executing in a Secure state.
- Non-secure physical PARTID space. This space is accessed when a Requester is executing in a Non-secure state.

PEs and some other Requesters have optional virtual PARTID spaces:

- Non-secure virtual PARTID space. This space exists only when the PE has the MPAM virtualization option implemented and enabled for the current EL.
- Secure virtual PARTID space. This space exists only when the PE has the MPAM virtualization option implemented and enabled for the current EL.

MPAM for RME provides two additional PARTID spaces:

- Realm PARTID space.
- Root PARTID space.

The PARTID spaces are parallel to the physical address spaces, but the PARTID space is not determined by the physical address space of an access. The PARTID space is determined by the Security state from which a memory access is made. Other factors, such as translation configuration, affect the physical address space of a memory access.

Table 3-2 Primary PARTID space for each Exception level and Security state in RME

Security state	EL3	EL2	EL1	EL0
Non-secure	n/a	Non-secure PARTID space	Non-secure PARTID space	Non-secure PARTID space
Secure	n/a	Secure PARTID space	Secure PARTID space	Secure PARTID space
Realm	n/a	Realm PARTID space	Realm PARTID space	Realm PARTID space
Root	Root PARTID space	n/a	n/a	n/a

Note

The primary PARTID space is determined only by the Security state and the Exception level.

Except for Non-secure state, if the PE supports the *Alternative PARTID spaces and selection* on page 6-87 PARTID space MPAM feature, an alternative PARTID space can be made available for software to use. See *Alternative PARTID spaces and selection* on page 6-87.

MPAM_NS indicates the PARTID space of a physical PARTID. When MPAM_NS is 0 it indicates the Secure physical PARTID space. When MPAM_NS is 1 it indicates the Non-secure physical PARTID space.

For RME, the MPAM_NS component of the MPAM information bundle is redefined to be a 2-bit value, MPAM_SP. The value of MPAM_SP[1:0] is given in Table 3-3 on page 3-37.

Table 3-3 : Encoding of 2-bit MPAM_SP

MPAM_SP[1:0]	MPAM PARTID space
`0b00`	Secure PARTID space
`0b01`	Non-secure PARTID space
`0b10`	Root PARTID space
`0b11`	Realm PARTID space

Each PARTID space has a maximum PARTID set by the implementation. 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 maximum PARTID implemented in each physical PARTID space. 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.

3.4.1 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 MPAMn_ELn.MPAMEN == 0, where n is the highest Exception level implemented. 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.

The PARTID error conditions in a PE are described in *MPAM errors and default ID generation* on page 6-85.

Note

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

3.4.2 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 PMG error conditions in a PE are described in *MPAM errors and default ID generation* on page 6-85.

Note

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

Chapter 4
Memory System Propagation of MPAM Information

This chapter contains the following sections:

4.1 Introduction

This section is *normative*.

The MPAM information bundle is propagated through the memory system components, or MSCs, that have MPAM resource controls or monitoring. The MPAM information bundle is described in *Introduction* on page 3-34.

MPAM information propagates in the direction of requests from Requesters towards terminating Completer components. This is the downstream direction. The upstream direction is from Completers towards Requesters.

The propagation behavior in the memory system depends on the function of the part of the memory system. Each MSC must implement at least one of the following behaviors:

- *Requester components* on page 4-41.
- *Terminating Completer components* on page 4-42.
- *Intermediate Completer-Requester components* on page 4-43.
- *Request buffering* on page 4-44.
- *Cache memory* on page 4-45.
- *MPAM for RME propagation of MPAM_SP with requests* on page 4-46.

If an MSC has no downstream components that use MPAM information, the MSC is not required to propagate MPAM information.

4.2 Requester components

Requesters must label all requests to downstream MSCs with MPAM information.

A Requester 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 Requester. Arm recommends that non-PE Requesters needing to use MPAM facilities specify a mechanism for determining the PARTID, PMG, and MPAM_NS of requests.
- *Arm System Memory Management Unit Architecture Specification, SMMU architecture versions 3.0, 3.1 and 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.
- *Arm Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0* specifies MPAM information generation on memory system accesses originated by the GIC to its tables in memory.

If a Requester 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, then these values must be driven to a default value, whether they are in the Non-secure physical PARTID space or in the Secure physical PARTID space.

See also *Requesters without MPAM support* on page 5-53.

4.3 Terminating Completer components

A terminating Completer receives requests from upstream Requesters but does not communicate the requests to a downstream Completer. Instead, the terminating Completer services the requests. A terminating Completer 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 Completer, 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 Completer that has MPAM support.

4.4 Intermediate Completer-Requester components

Intermediate MSCs have both one or more Completer interfaces and one or more Requester interfaces.

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

An intermediate component might terminate some requests from upstream locally without propagating the request to a downstream Requester 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 5-56 for requirements on cache memory behavior.

4.7 MPAM for RME propagation of MPAM_SP with requests

MPAM_SP is 2 bits in an MPAM for RME four-PARTID-space region. See *Four-space region* on page 5-59.

MPAM_SP must be propagated to all components within a four-space region.

MPAM_SP must be propagated to all bridges connecting a four-space region to a two-space region. See *Two-space region* on page 5-59 and *Systems with both two PARTID space and four PARTID space components* on page 5-59.

MPAM_SP must be propagated from all bridges connecting two-space regions to a four-space region.

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 Requesters, interconnects, caches, and memory controllers is described.

In this system model, a request:

- Begins at a Requester, such as a PE, I/O Requester, 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 5-56 and *Optional cache behaviors* on page 5-57.
- 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 Completer device responding to a request it receives.

Figure 5-1 on page 5-49 shows a simplified system model for the downstream flow, in the direction of requests from Requesters to Completers. In this figure, all objects implement an MSC except the PEs, I/O Requesters, and I/O Completers. PEs generate MPAM information from MPAM state in their System registers. I/O Requesters typically get their MPAM information when their requests pass through an SMMU.

The interconnects in Figure 5-1 on page 5-49 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.

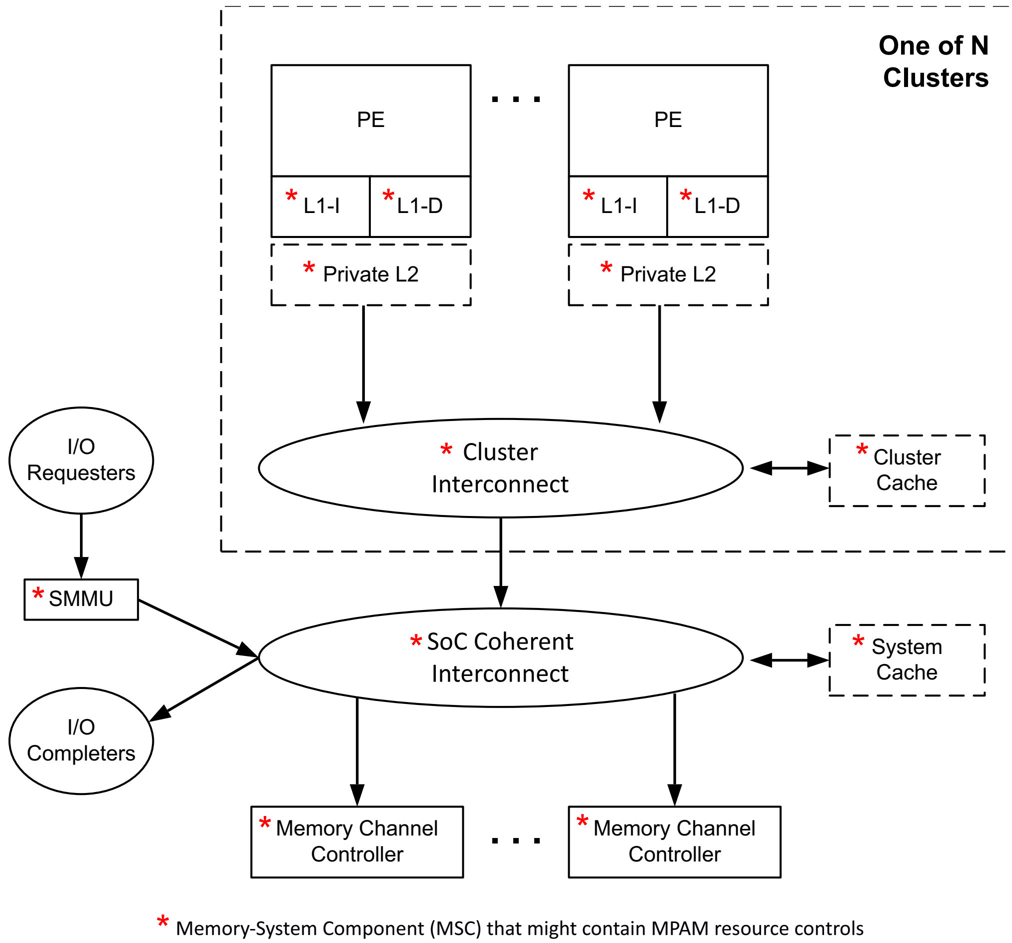


Figure 5-1 MPAM system model (downstream flow)

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 7-91) 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 7-95) 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 5-56 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 Requesters 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 3-38) 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 on page 5-54. 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.

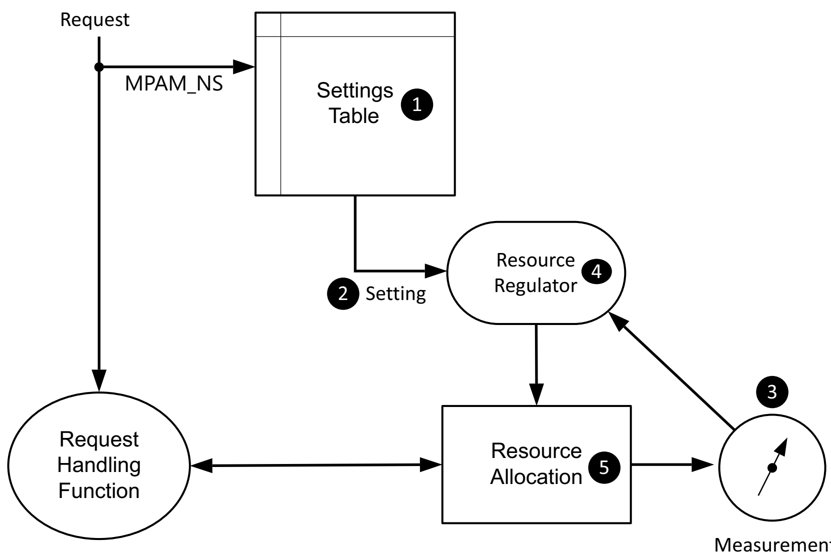


Figure 5-2 Model of MPAM resource partitioning controller

In Figure 5-2 on page 5-54, a request arrives from an upstream Requester 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 on page 5-54, 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 Completers, and they must transport MPAM information fields from Requester to Completer.

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 5-56 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 Requester 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 5-57).
- 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* on page 5-56), 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 Requesters.

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* on page 5-56) 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 9-177) 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 9-177) 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 monitors* on page 10-195). 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 9-179).
- Memory-bandwidth portion partitioning (*Memory-bandwidth portion partitioning* on page 9-179).
- Priority partitioning (internal) (*Priority partitioning* on page 9-183).
- Memory-bandwidth usage monitors (*Memory-bandwidth usage monitors* on page 10-193).

5.10 The MPAM for RME system

5.10.1 The MPAM for RME system

The MPAM for RME system supports RME PEs and at least one PE that supports both RME and MPAM for RME.

RME PEs support:
- Four Security states.
- Four physical address spaces.

A PE that supports RME and MPAM must also support MPAM for RME.

MPAM for RME requires support in the PE for:
- MPAM v1.1.
- Four MPAM PARTID spaces.
- MPAM alternative space (ALTSP) feature.

There are two possible space regions in an MPAM for RME system:
- A four-space region.
- A two-space region.

Like other MPAM systems, MPAM for RME can also contain non-MPAM components and subsystems. See *Non-MPAM components* on page 5-64.

Four-space region

This type of region:
- Contains one or more RME application PEs.
- Contains caches associated with those PEs.
- Contains cache-coherent interconnect among those PEs.
- Supports four MPAM PARTID spaces.

All components in a four-space region must support and use four PARTID spaces. If a component that can support four PARTID spaces is in a two-space region, then only those two PARTID spaces can be used.

Two-space region

This type of region contains a single two-space MPAM component or many two-space MPAM components connected as a subsystem through a two-space interconnect component. This component connects to the four-space region using a bridging scheme.

Two-space MPAM components support two PARTID spaces. These are compatible with MPAM v1.0 and MPAM v1.1 but lack support for the Root and Realm PARTID spaces.

Two-space MPAM components can be used in an MPAM for RME system, but with some loss of functionality and with some complication to the MPAM software.

Systems with both two PARTID space and four PARTID space components

When two-space MPAM components are included in a four PARTID space system, all four-space MPAM components receive requests from any four PARTID space Requesters with all four states propagated to the four-space components.

If the propagation of the four PARTID spaces in the MPAM information labels is blocked by two-space components between any four-space Requester and any four-space Completers, the interface where the four PARTID spaces are reduced to two PARTID spaces is the boundary to a two-space region and must reduce the MPAM_SP to MPAM_NS using a bridge. The Completer is part of a two-space region and uses only two PARTID spaces even though it supports four.

Figure 5-3 on page 5-60 shows a system with a large four-space region with support for four PARTID spaces and a smaller two-space region. The boxes labeled *2 to 4* and *4 to 2* are bridges chosen from *Bridging between four-space and two-space regions* on page 5-61.

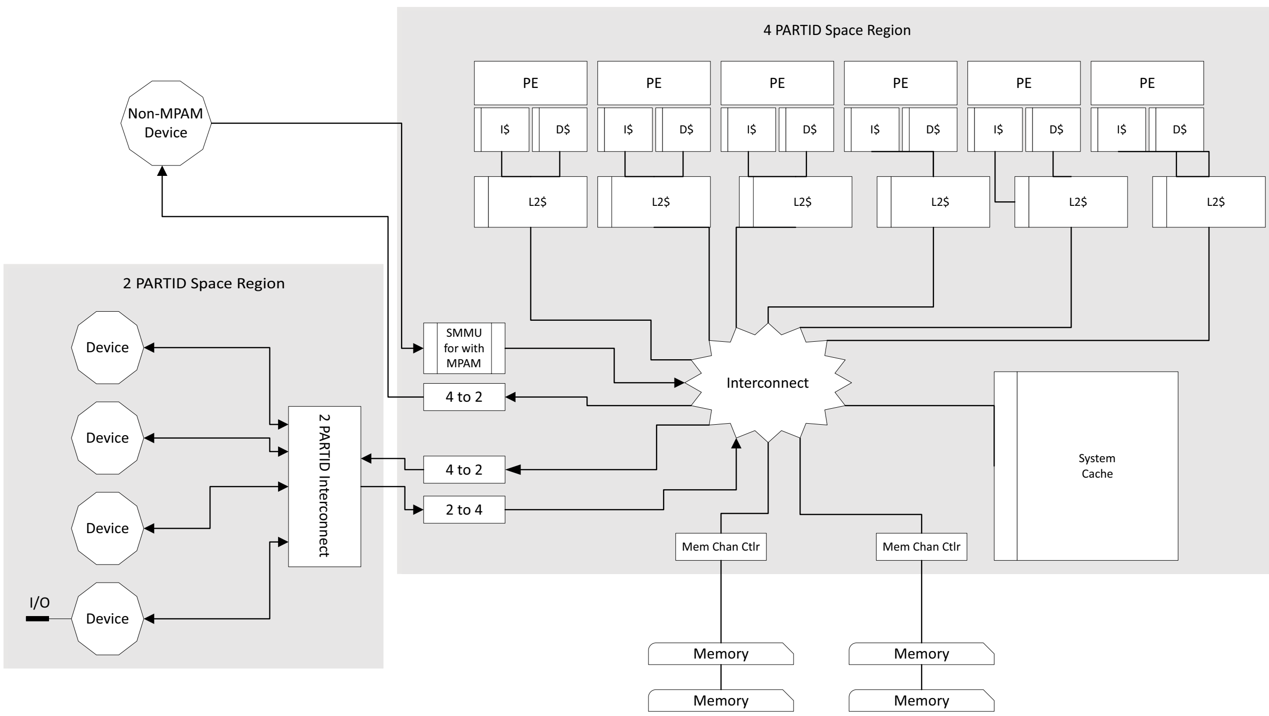


Figure 5-3 Example system with a large four PARTID space region and small two PARTID space regions

Figure 5-4 on page 5-61 shows a system with a small four-space region and a large two-space region. In this case the bridges are not shown. Here the PEs can use the ALTSP feature to produce two PARTID space requests without the need for bridging logic, using just the static bridge of the Completer. See *Alternative PARTID space and PARTID virtualization* on page 6-89 and *Fixed space mapping at a Completer* on page 5-64.

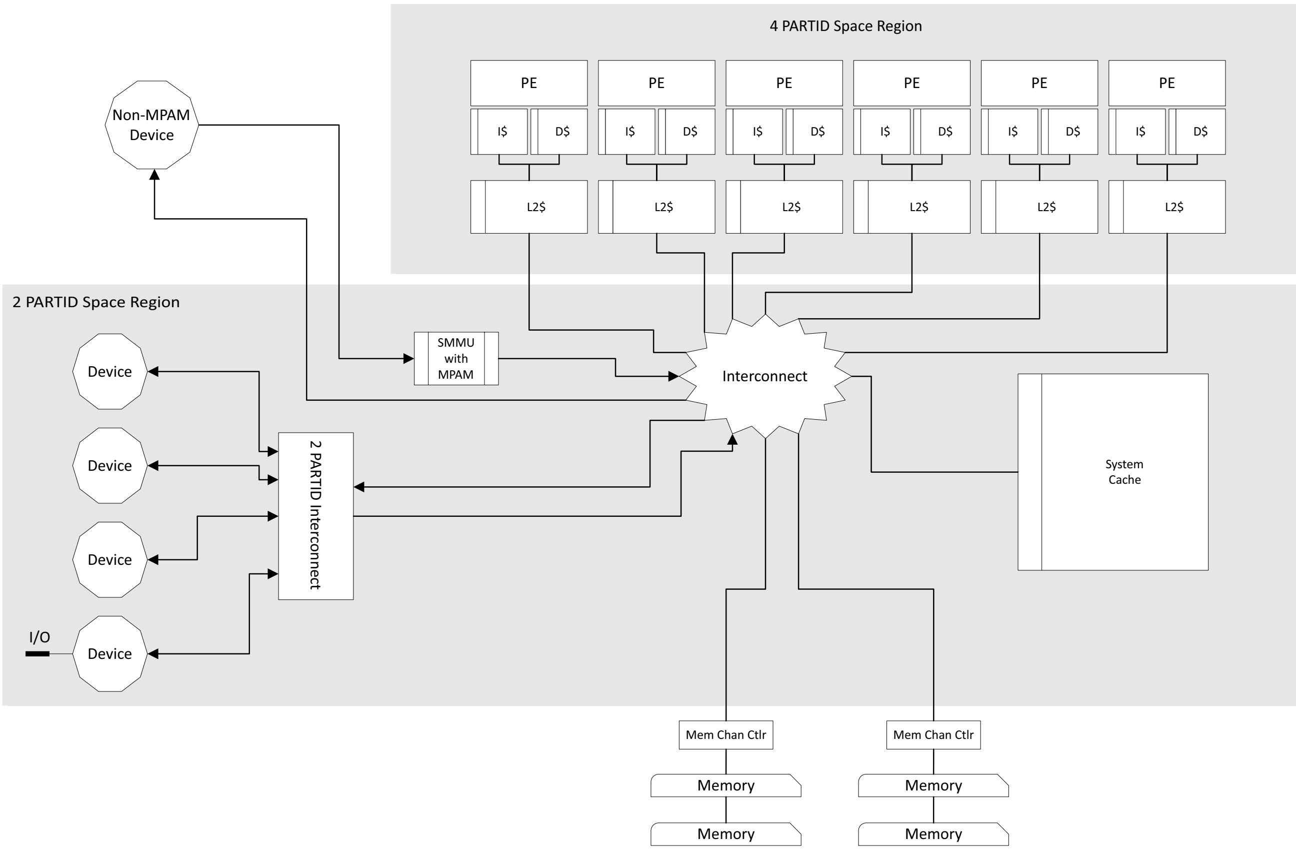


Figure 5-4 Example system with a small four PARTID space region and a large two PARTID space region

Requirements on bridges

The requirements on bridges are:

- The physical address space of a request must not be altered by bridging or other mechanisms.
- Bridging requests that use the Secure PARTID space must not be altered to use a different PARTID space.
- Bridging requests that use the Non-secure PARTID space must not be altered to use a different PARTID space.

5.10.2 Bridging between four-space and two-space regions

This section is *informative*.

Bridges are needed at the boundary between a four-space region and a two-space region. This section presents examples of bridging from two PARTID space Requesters to four PARTID space Completers and in the other direction from four PARTID space Requesters to two PARTID space Completers. Bridging schemes other than the examples given in this section can also be implemented.

Two-Space Requesters

When a two-space MPAM Requester is upstream from a four-space MSC, the Requester's MPAM labels must have the MPAM_NS field expanded to the 2-bit MPAM_SP[1:0] while satisfying the requirements in *Requirements on bridges* on page 5-61.

When bridging from a two-space region to a four-space region, Arm recommends a static mapping using the fixed MPAM_NS expansion.

Fixed MPAM_NS Expansion at a Requester

The fixed MPAM_NS expansion scheme transforms the MPAM_NS field to 2-bit MPAM_SP[1:0] field according to Table 5-1 on page 5-62:

Table 5-1 Two-space Requester to four-space fixed expansion scheme

Two-space MPAM_NS Input	Four-space MPAM_SP[1:0] Output
0b0 (Secure PARTID space)	0b00 (Secure PARTID space)
0b1 (Non-secure PARTID space)	0b01 (Non-secure PARTID space)

The fixed expansion scheme preserves the PARTID space across the mapping.

Two-Space Completers

When a two-space MPAM Completer is downstream from a four-space Requester, the Requester's MPAM labels must have the MPAM_SP field reduced to form the 1-bit MPAM_NS while satisfying the requirements in *Requirements on bridges* on page 5-61. The reduction function may be static or dynamic.

Note

Arm makes no recommendation for which method to use for bridging between the four-space region of a system that has four PARTID spaces and a two-space region that supports two PARTID spaces. All known methods affect the system operation in ways that could cause difficulties for software.

Control over monitoring of Root and Realm PARTID space requests bridged to Secure or Non-secure PARTID space

A NO_MON flag is used in some of the examples to indicate that the transaction must not be monitored by MPAM monitors or other system performance monitors. This capability improves the security by limiting or preventing the system-level activities of a Realm from being collected in monitors accessible from the Non-secure physical address space or Secure physical address space.

The choice of not monitoring some transactions is not available on true two-space components. Support for the ability to mark requests with the NO_MON flag would likely require modifying the two-space component.

The examples that follow show a small number of recommended choices for including two-space MPAM Completer MSCs that do not have four-space MPAM support in an RME system. Example 5-1 on page 5-62 is the most desirable option, but requires extensive work in that it requires a redesign of the MSC. Example 5-4 on page 5-63 requires the least effort but is also the least desirable option:

Example 5-1 Alter the two-space MSC to support 4 PARTID spaces

This is the recommended option. However, it requires work to redesign the MSC. See *Four-space MSC* on page 8-170 for how this is implemented.

Example 5-2 Alter the two-space MSC to support a programmable mapping of 4 PARTID spaces to 2 PARTID spaces

Alter the two-space MSC to support a programmable mapping of 4 PARTID spaces to 2 PARTID spaces with additional control over whether each of the Root and Realm PARTID spaces can be monitored. See *Programmable PARTID space mapping within a Completer* on page 5-63.

Example 5-3 Connect the two-space MSC through a programmable PARTID-space mapping component

---

Connect the two-space MSC through a programmable PARTID-space mapping component, or shim. See *Space mapping external to an MSC* on page 5-63.

This gives no control of whether the Root or Realm space can be monitored after being mapped into Secure or Non-secure.

Example 5-4 Connect the two-space MSC to be driven only from MPAM_SP[0]

Connect the two-space MSC so that the single-bit MPAM_NS input of the two-space MSC is driven only from MPAM_SP[0]. See *Fixed space mapping at a Completer* on page 5-64.

Programmable PARTID space mapping within a Completer

See Example 5-2 on page 5-62.

A programmable MPAM PARTID space mapping can be performed for a MSC with an PARTID space mapping built into the component. The PARTID space mapper accepts the request with 4 MPAM spaces, maps requests with MPAM_SP of Root or Realm to one of the Secure or Non-secure PARTID spaces and passes it on to the two-space MSC.

The programmable mapper can also produce a flag that indicates the two-space MSC should not perform MPAM monitoring of the request. See *Control over monitoring of Root and Realm PARTID space requests bridged to Secure or Non-secure PARTID space* on page 5-62.

The request mapper programming register is MAP4SPTO2SP. It has the fields shown in Table 5-2 on page 5-63:

Table 5-2 Request mapper programming register (MAP4SPTO2SP) fields

Field bits	Field name	Description
15	Rt_outPARTID_space	If a request has a Root PARTID, the output PARTID uses this bit for MPAM_NS.
14	Rt_NO_MON	If the request has a Root PARTID, output this bit as the NO_MON flag.
7	Rl_outPARTID_space	If a request has a Realm PARTID, the output PARTID uses this bit for MPAM_NS.
6	Rl_NO_MON	If the request has a Realm PARTID, output this bit as the NO_MON flag.

The MAP4SPTO2SP register must only be accessible in the Root physical address space.

Space mapping external to an MSC

See Example 5-3 on page 5-63.

A two-space Completer can be connected using a small component external to the MSC that implements a programmable four-space to two-space mapping similar to MAP4SPTO2SP. See Table 5-3 on page 5-64:

Table 5-3 *Space mapping external to the MSC* MAP4SPTO2SP fields

Field bits	Field name	Description
15	Rt_outPARTID_space	If a request has a Root PARTID, the output PARTID uses this bit for MPAM_NS.
14	Rt_NO_MON	If the request has a Root PARTID, output this bit as the NO_MON flag.
7	Rl_outPARTID_space	If a request has a Realm PARTID, the output PARTID uses this bit for MPAM_NS.
6	Rl_NO_MON	If the request has a Realm PARTID, output this bit as the NO_MON flag.

The external mapping register must only be accessible in the Root physical address space.

If the two-space MSC does not have any way to accept the NO_MON flag at the request input, the NO_MON flag is not used. Two-space MSCs are not required to support a NO_MON input.

Fixed space mapping at a Completer

See Example 5-4 on page 5-63.

The fixed MPAM_SP reduction scheme transforms MPAM_SP into a 1-bit MPAM_NS according to Table 5-4 on page 5-64:

Table 5-4 Four-space to two-space static reduction scheme

Four-space MPAM_SP Input	Two-space MPAM_NS Output
0b00 (Secure PARTID space)	0b0 (Secure PARTID space)
0b01 (Non-secure PARTID space)	0b1 (Non-secure PARTID space)
0b10 (Root PARTID space)	0b0 (Secure PARTID space)
0b11 (Realm PARTID space)	0b1 (Non-secure PARTID space)

5.10.3 Non-MPAM components

Non-MPAM components do not have the ability to make requests with non-zero MPAM information or to use MPAM information when completing requests. They also do not propagate MPAM information to downstream MSCs.

Non-MPAM Requesters

Note

Arm strongly recommends that an SMMU for RME, see *Arm® System Memory Management Unit Architecture Specification, SMMU architecture* (ARM IHI 0070) with the SMMUv3.2-MPAM feature, see *Arm® Realm Management Extension (RME), for SMMUv3 Arm® System Memory Management Unit Architecture Supplement* (ARM IHI 0094) is used to add MPAM information to requests from non-MPAM Requesters.

Requesters attached to an SMMU for RME are only associated with the Secure and Non-secure states, and therefore use two of the four PARTID spaces.

NoStreamID requesters attached to an SMMU for RME might issue transactions to Root or Realm physical address space. For these accesses it is permitted to use Secure and Non-secure PARTID spaces respectively.

Non-MPAM Completers

Completers that have no support for the MPAM information accompanying requests should be interfaced to the system by dropping MPAM information from the requests.

A non-MPAM Completer limits the topology of MPAM in the system because it does not propagate MPAM information to MPAM components downstream. See *Systems with both two PARTID space and four PARTID space components* on page 5-59.

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 PARTID space MPAM_SP, if RME is implemented, and MPAM_NS if not, labels for memory-system requests are controlled by software running at the current Exception level 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.

Note

For information on the presence of MPAM functionality in a PE, see *MPAM versions for PEs* on page 1-22.

6.2 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 6-1 on page 6-69.

Table 6-1 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 on page 6-69. These registers control MPAM PARTID and PMG, as shown in Table 6-2 on page 6-70 and *Summary of System registers* on page 7-94 and *System register descriptions* on page 7-95.

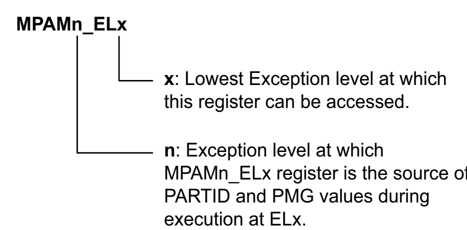


Figure 6-1 MPAM System register name syntax

Table 6-2 on page 6-70 shows the PE MPAM System registers. The table does not include the following System registers: MPAMIDR_EL1, MPAMVPMn_EL2, MPAMVPMV_EL2, MPAMHCR_EL2.

Table 6-2 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 6-3 on page 6-71 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 6-3 Selection of MPAMn_ELx System register for MPAM generation

Current Exception level	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

6.3 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 6-3 on page 6-71. 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 6-3 on page 6-71. 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.

6.3.1 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.

6.3.2 Accesses by enhanced support for nested virtualization

In Armv8.4, 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 6-82.

6.3.3 Accesses by statistical profiling extension

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.

6.3.4 Translation table accesses by AT instructions

Accesses to translation tables by AT instructions are given the MPAM information specified for translation table accesses by a data load instruction that is issued from the Exception level that the AT instruction was executed from. The stage and Exception level specified in the AT instructions do not affect the MPAM information to use.

6.3.5 MPAM information for Granule Protection Table access

In MPAM for RME, accesses to the Granule Protection Table (GPT) use MPAM information according to the current execution Exception Level and Security state and the type of access. See *MPAM information for Granule Protection Table access* on page 6-90.

6.4 Security

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

The generation of PARTID and PMG for a memory-system request is the same at an ELn in any Security state for the same n. The difference is that requests have the PARTID space derived from the Security state indicated on MPAM_NS by PEs that do not implement RME and on MPAM_SP by PEs that implement RME.

MPAM security behavior in MSCs is covered in *Security in MSCs* on page 8-162.

6.4.1 Secure and Non-secure PARTID space

In a two-space and four-space PE, generation of Secure PARTIDs are governed by the following Secure MPAM PARTID space rules, described in *PARTID spaces and properties* on page 3-37:

- PARTIDs in the Secure PARTID space are communicated with MPAM_NS as `0b0` when RME is not implemented or with MPAM_SP as `0b00` if RME is implemented.
- PARTIDs in the Non-secure PARTID space are communicated with MPAM_NS as `0b1` when RME is not implemented or MPAM_SP as `0b01` when RME is implemented.
- When in Secure state:
  - If the MPAM version is v1.0 or greater, MPAM_NS is always `0b0`.
  - If the MPAM version is less than v1.0, MPAM_NS might be `0b0` or `0b1`. For more information, see *SDEFLT and FORCE_NS settings to control Secure MPAM PARTID use* on page 6-73.
  - In MPAM for RME, MPAM_SP in the Secure state can be either `0b00` or `0b01`. For more information, see *MPAM for RME PE generation of MPAM information* on page 6-87.

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 6-76.

6.4.2 Relationship of PARTID space and physical address space

The PARTID space and the physical address space of a memory transaction initiated by a PE are both based on the Security state, either the current Security state of the PE or in some limited situations, the Security state of the owning Exception level.

The primary PARTID space is always based on the Security state as given in *Primary PARTID space for each Exception level and Security state in RME* on page 3-37. However, the PARTID space may be changed by MPAM3_EL3.FORCE_NS in MPAM v0.1 or by the alternative PARTID space MPAM feature in MPAM for RME. See *Settings to control Secure MPAM PARTID use in MPAM v0.1 implementations* on page 6-74 and *Settings to control MPAM PARTID use in MPAM for RME* on page 6-75.

The physical address space is also based on the PE Security state but may be altered by the MMU in limited situations. For more information, see *Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile (ARM DDI 0487), Control of Secure or Non-secure memory access.*

6.4.3 SDEFLT and FORCE_NS settings to control Secure MPAM PARTID use

The settings to control the use of Secure MPAM PARTIDs vary depending on the version of MPAM implemented. MPAMv1.0 does not implement MPAM3_EL3.{SDEFLT, FORCE_NS} and so the settings are as described in *Secure and Non-secure PARTID space* on page 6-73. The Secure MPAM PARTID use settings for MPAMv1.1 and MPAMv0.1 are detailed in:

- *Settings to control Secure MPAM PARTID use in MPAM v1.1 implementations* on page 6-74

- *Settings to control Secure MPAM PARTID use in MPAM v0.1 implementations* on page 6-74

Settings to control Secure MPAM PARTID use in MPAM v1.1 implementations

The MPAM3_EL3.SDEFLT control enables partial support of Secure PARTIDs as in Table 6-4 on page 6-74.

Table 6-4 Behaviors of MPAM3_EL3.SDEFLT in MPAMv1.1 implementations

MPAM3_EL3.SDEFLT	Behavior: Non-secure state	Behavior: Secure state
`0b0`	Compatible with MPAMv1.0. PARTID is in the Non-secure PARTID space. PARTID and PMG are generated from MPAMn_ELx registers.	Compatible with MPAMv1.0 PARTID is in the Secure PARTID space. PARTID and PMG are generated from MPAMn_ELx registers.
`0b1`		PARTID is in the Secure PARTID space. PARTID and PMG are generated as the default PARTID and default PMG.

Settings to control Secure MPAM PARTID use in MPAM v0.1 implementations

The MPAM3_EL3.SDEFLT and MPAM3_EL3.FORCE_NS controls enable partial support of Secure PARTIDs as in Table 6-5 on page 6-74.

Table 6-5 Behaviors of MPAM3_EL3.SDEFLT and MPAM3_EL3.FORCE_NS

MPAM3_EL3		Behavior	
SDEFLT	**FORCE_NS**	**Non-secure state**	**Secure state**
`0b0`	`0b0`	Compatible with MPAMv1.0. PARTID is in the Non-secure PARTID space. PARTID and PMG are generated from MPAMn_ELx registers.	Compatible with MPAMv1.0. PARTID is in the Secure PARTID space. PARTID and PMG are generated from MPAMn_ELx registers.
	`0b1`		PARTID is in the Non-secure PARTID space. PARTID and PMG are generated from MPAMn_ELx registers.
`0b1`	`0b0`		PARTID is in the Secure PARTID space. PARTID and PMG are generated as the default PARTID and default PMG.
	`0b1`		PARTID is in the Non-secure PARTID space. PARTID and PMG are generated as the default PARTID and default PMG.

If an implementation has MPAMIDR_EL1.HAS_FORCE_NS enabled, the implementation has two options:

- Secure PARTIDs are not implemented. MPAM3_EL3.FORCE_NS is RAO/WI.
- MPAM3_EL3.FORCE_NS can be written by software. MPAM3_EL3.FORCE_NS is RW.

Software can discover which of these options is implemented by testing whether MPAM3_EL3.FORCE_NS is writable to zero.

Settings to control MPAM PARTID use in MPAM for RME

When RME is implemented, the PE has four Security states. The controls that affect PARTID space and value involve alternative PARTID space selection with fields in MPAM3_EL3 and MPAM2_EL2 affecting the behavior in all ELs and Security states. The MPAM3_EL3.SDEFLT control uses only PARTID 0 for all PARTIDs generated in the Secure Security state. The behaviors are also dependent on whether the feature ALTSP is used (see *Alternative PARTID spaces and selection* on page 6-87).

These behaviors are described in Table 6-6 on page 6-75. The Alternative space selected column indicates whether the alternative PARTID space is selected instead of the primary PARTID space. If alternative PARTID space MPAM feature is not implemented, the alternative PARTID space is never selected.

Table 6-6 Behaviors of MPAM3_EL3 and MPAM2_EL2 controls on PARTID use in MPAM for RME

MPAM3_EL3.SDEFLT	**Alternative space selected**	**Root state behavior**	**Secure state behavior**	**Realm state behavior**	**Non-secure state behavior**
0	No	EL3 PARTID generated from MPAM3_EL3 in the Root PARTID space.	EL2-EL0 PARTID generated normally from MPAMn_ELx registers in the Secure PARTID space.	EL2-EL0 PARTID generated normally from MPAMn_ELx registers in the Realm PARTID space.	EL2-EL0 PARTID generated normally from MPAMn_ELx registers in the Non-secure PARTID space.
1	No	EL3 PARTID generated from MPAM3_EL3 in the Root PARTID space.	EL2-EL0 PARTID generated as PARTID 0 in the Secure PARTID space.	EL2-EL0 PARTID generated normally from MPAMn_ELx registers in the Realm PARTID space.	EL2-EL0 PARTID generated normally from MPAMn_ELx registers in the Non-secure PARTID space.
0	Yes	EL3 PARTID generated from MPAM3_EL3 in the Secure or Non-secure PARTID space.	EL2-EL0 PARTID generated normally from MPAMn_ELx registers in the Non-secure PARTID space.	EL2-EL0 PARTID generated normally from MPAMn_ELx registers in the Non-secure PARTID space.	EL2-EL0 PARTID generated normally from MPAMn_ELx registers in the Non-secure PARTID space.
1	Yes	EL3 PARTID generated from MPAM3_EL3 in the Secure or Non-secure PARTID space.	EL2-EL0 PARTID generated as PARTID 0 in the Non-secure PARTID space.	EL2-EL0 PARTID generated normally from MPAMn_ELx registers in the Non-secure PARTID space.	EL2-EL0 PARTID generated normally from MPAMn_ELx registers in the Non-secure PARTID space.

6.5 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 7-152.
- 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 and MPAM2_EL2.TIDR bits. 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. See *Trap accesses to EL2 and EL1 System registers* on page 6-80.

  Note

  MPAM2_EL2.TIDR is present when MPAM v0.1 or MPAM v1.1 are implemented and MPAMIDR_EL1.HAS_TIDR is 1.

- 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 6-85.
- 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 6-77.

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.

6.5.1 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.

When MPAMEN is 0, the default physical PARTID must be generated for all memory-system requests.

Virtual PARTID spaces can be independently enabled for MPAM0_EL1 and MPAM1_EL1 in MPAMHCR_EL2. See Table 6-3 on page 6-71. 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 6-79.

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 3-38.

6.5.2 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 6-7 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 6-8 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.

6.5.3 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 6-81).

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.

6.5.4 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.

6.5.5 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 to MPAM registers from EL2 and EL1.

Virtualizing MPAMIDR_EL1

EL2 software can force accesses to MPAMIDR_EL1 to trap to EL2 by setting MPAMHCR_EL2.TRAP_MPAMIDR_EL1 == 1 or MPAM2_EL2.TIDR == 1. By trapping MPAMIDR_EL1, an EL2 hypervisor can provide an emulated value for MPAMIDR_EL1 to the EL1 software.

Note

MPAM2_EL2.TIDR is present when MPAMIDR_EL1.HAS_TIDR is 1. Arm recommends that when MPAM v0.1 or MPAM v1.1 are implemented, MPAMIDR_EL1.HAS_TIDR is set to 1 and MPAM2_EL2.TIDR is implemented.

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 all lower Exception levels to EL3 by setting MPAM3_EL3.TRAPLOWER == 1.

TRAPLOWER protects against misuse of the MPAM state registers when EL2 software 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.

6.6 MPAM AArch32 interoperability

This section is *normative*.

MPAM System registers are not accessible from AArch32, so the MPAM PARTIDs and PMGs for any Exception level that uses AArch32 state must be set up by a higher Exception level that uses AArch64 state.

6.7 Support for nested virtualization

This section is *normative*.

Armv8.3 Extensions added FEAT_NV for nested virtualization, and *Armv8.4 Extensions* added FEAT_NV2 to the nested virtualization support. This section describes the support of MPAM with these extensions.

6.7.1 Nested virtualization extension

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

Table 6-9 on page 6-82 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 6-9 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.

6.7.2 Enhanced nested virtualization extension

Armv8.4 Extensions introduced FEAT_NV2, 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 the FEAT_NV and FEAT_NV2 extensions 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 6-10 on page 6-83 as described in *Armv8.4 Extensions*.

Table 6-10 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 6-11 on page 6-83 (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 6-9 on page 6-82 as described in *Armv8.4 Extensions*.

Table 6-11 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`

Table 6-11 Enhanced nested virtualization offsets of System registers (NV2 == 1, NV1 == 1 and NV == 1) (continued)

Register Name	Offset
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 6-10 on page 6-83 if HCR_EL2.NV1 == 0 or in Table 6-11 on page 6-83 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.

6.7.3 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.

6.8 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 3-38.

6.8.1 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.

6.8.2 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.

6.8.3 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* on page 6-85) 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 3-38.

6.8.4 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 6-12 on page 6-85.

Table 6-12 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

6.8.5 MPAM3_EL3.SDEFLT and MPAM generation errors

When executing in Secure state, MPAM3_EL3.SDEFLT sets the MPAM generation to produce only zero for PARTIDs. The default PARTID is always valid, so PARTID Out-of-range errors cannot occur in Secure state when MPAM3_EL3.SDEFLT is 1.

Note

MPAM3_EL3.MPAMEN and MPAM3_EL3.SDEFLT have a similar function. However, when MPAM3_EL3.MPAMEN is 0 in Secure or Non-secure state:

- MPAM generation produces only zero for the physical PARTID in all memory-system requests.
- Virtual PARTID mapping is not performed.
- PARTIDs cannot generate Out-of-range errors.

When MPAM3_EL3.SDEFLT is 1, PMG is always 0, and always valid. PMG generation is not virtualized.

6.8.6 MPAM3_EL3.FORCE_NS and MPAM generation errors

MPAM3_EL3.FORCE_NS is only present in MPAM v0.1. When in Secure state, MPAM3_EL3.FORCE_NS changes the MPAM generation in the PE so that MPAM_NS is set to 1 rather than 0. This means that only Non-secure MPAM information will accompany memory system requests from a PE, so MSCs will receive Non-secure PARTIDs from those requests.

FORCE_NS does not change the way that the value of the PARTID is produced, only whether the generated PARTID is a Secure PARTID or a Non-secure PARTID. This means that generation of the physical PARTID and PMG for the MPAM information to label memory system requests are unchanged by FORCE_NS. The generation of MPAM information in the PE can produce any of the MPAM generation error behaviors described in *MPAM errors and default ID generation* on page 6-85.

6.9 MPAM for RME PE generation of MPAM information

A PE that implements FEAT_RME has the capability to execute in the Realm and Root Security states and to generate accesses to the Realm and Root physical address spaces.

6.9.1 PE and MPAM

A PE that implements FEAT_RME must generate the PARTID space according to the Security state from which the memory system request is made.

Each Security state has a primary PARTID space named for that Security state as shown in Table 6-13 on page 6-87.

The alternative PARTID space MPAM feature, ALTSP, allows an alternative PARTID space to be used in each Security space rather than the primary PARTID space. See *Alternative PARTID spaces and selection* on page 6-87.

Table 6-13 MPAM_SP encoding for each PARTID space

PARTID Space	MPAM_SP[1:0]
Non-secure PARTID space	0b01
Secure PARTID space	0b00
Realm PARTID space	0b11
Root PARTID space	0b10

Support by the PE for the four PARTID spaces is identified in MPAMIDR_EL1. In a PE that implements RME and MPAM, MPAMIDR_EL1.SP4 must be 1.

6.9.2 Alternative PARTID spaces and selection

The Alternative PARTID Space feature, ALTSP, defines alternative PARTID spaces for each of the Security states.

MPAM3_EL3 and MPAM2_EL2 have fields to control whether the primary or alternative PARTID space is used at those Exception levels and lower Exception levels.

The ALTSP feature permits the selection of either the primary PARTID space or the alternative PARTID space for PARTIDs in the MPAMn_ELx registers. The primary and alternative PARTID spaces for each Security state are shown in Table 6-14 on page 6-87. The primary PARTID space is shown, where the PARTID space name is the same as the Security state.

Table 6-14 Primary and alternative PARTID spaces

Security state	Primary PARTID space	Alternative PARTID Space
Non-secure	Non-secure PARTID space	Same
Secure	Secure PARTID space	Non-secure PARTID space
Realm	Realm PARTID space	Non-secure PARTID space
Root	Root PARTID space	Secure or Non-secure PARTID space

The choice of the alternative space for Root is made in MPAM3_EL3 in the RT_ALTSP_NS field:

- 0b1 selects the Non-secure PARTID space as the alternative PARTID space for the Root Security state.
- 0b0 selects the Secure PARTID space as the alternative PARTID space for the Root Security state.

The ALTSP feature controls do not affect the PARTID space when used in the Non-secure state. The Non-secure PARTID space is always used in the Non-secure Security state.

See MPAM3_EL3 and MPAM2_EL2 for details of these controls. The ALTSP feature is identified in MPAMIDR_EL1.HAS_ALTSP.

Selection of primary or alternative PARTID space when executing at EL3

When executing at EL3, the PE is in the Root Security state.

The selection of primary or alternative PARTID space for memory system requests generated in the Root Security state is controlled by these bits in MPAM3_EL3:

- RT_ALTSP_NS sets whether the alternative PARTID space in the Root Security state is the Non-secure PARTID space or the Secure PARTID space.
- ALTSP_EL3 sets whether memory system requests generated from EL3 use the alternative PARTID space or the primary PARTID space. The selected PARTID space is used for all accesses that use MPAM3_EL3.PARTID_I or MPAM3_EL3.PARTID_D.

These two bits combine to give three combinations for PARTID space used for accesses from EL3 in the Root state.

Table 6-15 EL3 PARTID space selection

MPAM3_EL3.RT_ALTSP_NS	MPAM3_EL3.ALTSP_EL3	PARTID space
x	0	Root PARTID space
0	1	Secure PARTID space
1	1	Non-secure PARTID space

Selection of primary or alternative PARTID space when executing at EL2, EL1 and EL0

When executing at EL2, EL1 or EL0, the Security state can be one of Secure, Non-secure, or Realm. The current Security state for all Exception levels below EL3 is set in SCR_EL3 by the NS and NSE bits.

The Root firmware running in EL3 can either permit EL2 to control its own PARTID space and the PARTID space used by EL1 and EL0, or it can force the primary or alternative space to be selected for EL2, EL1, and EL0.

If EL3 is not forcing a selection on EL2, EL2 can select whether PARTIDs generated at EL2 use the primary or alternative PARTID space using MPAM2_EL2.ALTSP_EL2. When EL3 is not forcing a selection on EL2, EL2 can also select whether the primary or alternative PARTID space is used by EL1 and EL0.

EL3 forces a selection on all lower ELs by clearing MPAM3_EL3.ALTSP_HEN and setting MPAM3_EL3.ALTSP_HFC to force the alternative PARTID space or clearing ALTSP_HFC to force the primary PARTID space on all lower ELs.

When EL2 is implemented but is disabled for the Security state, the alternative PARTID space is selected for EL1 and EL0 when MPAM3_EL3.ALTSP_HEN is 0 and MPAM3_EL3.ALTSP_HFC is 1. Otherwise the primary PARTID space is selected.

The set of combinations for EL2 PARTID space selection are shown in Table 6-16 on page 6-88.

Table 6-16 EL2 PARTID space selection

MPAM3_EL3. ALTSP_HEN	MPAM3_EL3. ALTSP_HFC	MPAM2_EL2. ALTSP_EL2	EL2 PARTID space
0	0	x	Primary
0	1	x	Alternative
1	x	0	Primary
1	x	1	Alternative

The set of combinations for EL1 and EL0 PARTID space selection are shown in Table 6-17 on page 6-89. When EL2 is not implemented or when EL2 is implemented but not enabled for the Security state, read Table 6-17 on page 6-89 as if MPAM2_EL2.ALTSP_HFC is 0.

Table 6-17 EL1 and EL0 PARTID space selection

MPAM3_EL3. ALTSP_HEN	MPAM3_EL3. ALTSP_HFC	MPAM2_EL2. ALTSP_HFC	EL1 and EL0 PARTID space
0	0	x	Primary
0	1	x	Alternative
1	x	0	Primary
1	x	1	Alternative

Determining forced PARTID space in EL2, EL1 and EL0

In each of MPAM2_EL2 and MPAM1_EL1, the ALTSP_FRCD bit indicates that the alternative PARTID space has been forced on PARTIDs in MPAM2_EL2 and on PARTIDs in MPAM1_EL1 and MPAM0_EL1, respectively. Since EL1 and EL0 selection is always identical and EL1 controls PARTIDs in MPAM0_EL1, there is no need for a separate indication in MPAM0_EL1.

There is no means provided for software running in EL0 to determine whether it is using the primary or alternative PARTID space. The PARTID space selection at EL0 is the same as for the Exception level of the operating system that controls the EL0 application. That OS is at EL2 if the virtualization host extension, host mode, is being used as indicated when MPAMHCR_EL2.E2H and MPAMHCR_EL2.TGE are both set to 1. Otherwise, the controlling operating system is at EL1.

Alternative PARTID space and PARTID virtualization

Because the choice of primary or alternative PARTID spaces at EL1 and EL0 can be controlled from EL2 and because MPAM1_EL1.PARTID_I and MPAM1_EL1.PARTID_D are in the same PARTID space, EL2 can set up PARTID virtualization into the correct PARTID space for EL1.

Similarly, MPAM1_EL1.PARTID_I and MPAM0_EL1.PARTID_D are in the same PARTID space as the MPAM1_EL1 PARTIDs so that the PARTID virtualization setup for EL1 can also be used for EL0.

PARTID virtualization is enabled for MPAM1_EL1 PARTIDs by setting MPAMHCR_EL2.EL1_VPMEN and for MPAM0_EL1 PARTIDs by setting MPAMHCR_EL2.EL0_VPMEN. Setting up PARTID virtualization also requires EL2 software to program physical PARTIDs from the selected PARTID space into the MPAMVPM<n>_EL2 registers, and enable those translations in the MPAMVPMV_EL2 register.

ALTSP and FORCE_NS

ALTSP can have the same effect of forcing PARTIDs in the Secure state to be in the Non-secure PARTID space as MPAM3_EL3.FORCE_NS. ALTSP also provides controls for the Root and Realm Security state selection of PARTID space.

Note

ALTSP and FORCE_NS are conflicting MPAM features. The ALTSP feature is required and the FORCE_NS feature is prohibited in PEs that implement MPAM for RME.

ALTSP in Host mode at EL0

When a host OS running at EL2 executes an application, it expects the same behavior as if it was an EL1 OS running an application. The behaviors to support running a host OS at EL2 are controlled by HCR_EL2.E2H. The control bit HCR_EL2.TGE supports running an application of the host OS at EL0.

When running at EL0 in host mode, the EL2 selection of primary versus alternative PARTID space is used to govern the selection in EL0.

When HCR_EL2.E2H and HCR_EL2.TGE are both 1, the alternative PARTID space in EL0 is selected only if the alternative space would be selected in EL2. When either ofE2H or TGE is 0, the alternative PARTID space in EL0 is selected only if the alternative space would be selected in EL1.

6.9.3 MPAM information for Granule Protection Table access

In MPAM for RME, accesses to the Granule Protection Table (GPT) as a result of a data access, instruction access or translation table walk use the same MPAM information as the original access.

- A GPT access as the result of a data access uses PARTID_D and PMG_D for the current Exception level.
- A GPT access as the result of an instruction access uses PARTID_I and PMG_I for the current Exception level.
- A GPT access as the result of a translation table walk uses the PARTID_D and PMG_D for the current Exception level.
- A GPT access uses the PARTID space selected from the current Security state and current Exception level according to *Alternative PARTID spaces and selection* on page 6-87.

Chapter 7
System Registers

This chapter contains the following sections:

7.1 Overview

System registers are implemented in PEs and accessed using the MRS and MSR instructions.

7.2 Synchronization of System register changes

Direct writes to System registers are only guaranteed to be visible to indirect reads after a Context synchronization event, as described in the *Arm Architecture Reference Manual Armv8, for Armv8-A architecture profile*.

Writes to MPAM System registers 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 Exception level and used for generation of MPAM information at another Exception level, the change of Exception level 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 Exception level 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, 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 Armv8, for Armv8-A architecture profile* 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 must ensure that the prior memory accesses are completed before the update. No such System registers are defined here. 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. This means that MPAM information is not permitted to be cached in a TLB and used instead 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 on page 7-94 control the generation of PARTID and PMG by the PE, according to the Exception level and configuration of MPAM. See *Versions of the MPAM Extension* on page 1-22.

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

Configurations

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

Attributes

MPAM0_EL1 is a 64-bit register.

Field descriptions

63 48	47 40	39 32
RES0	PMG_D	PMG_I

31 16	15 0
PARTID_D	PARTID_I

Bits [63:48]

Reserved, RES0.

PMG_D, bits [47:40]

Performance monitoring group property for PARTID_D.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

PMG_I, bits [39:32]

Performance monitoring group property for PARTID_I.

The reset behavior of this field is:

- On a Warm reset, 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.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

PARTID_I, bits [15:0]

Partition ID for instruction accesses made from EL0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing 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 register 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) && MPAM3_EL3.TRAPLOWER == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EL2Enabled() && 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
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            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) && MPAM3_EL3.TRAPLOWER == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EL2Enabled() && 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
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            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

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

When EL2 is implemented and enabled in the current Security state, 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 implemented and enabled in the current Security state, 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 bit [63] is architecturally mapped to AArch64 System register MPAM3_EL3[63] when EL3 is implemented.

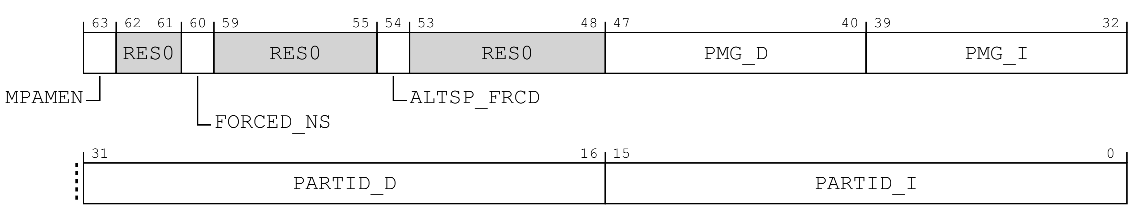
AArch64 System register MPAM1_EL1 bit [63] is architecturally mapped to AArch64 System register MPAM2_EL2[63] when EL3 is not implemented and EL2 is implemented.

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

Attributes

MPAM1_EL1 is a 64-bit register.

Field descriptions



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.

`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.

The reset behavior of this field is:

- On a Warm reset, this field resets to `0`.

Accessing this field has the following behavior:

- When EL3 is not implemented and EL2 is not implemented, access to this field is RW.
- Otherwise, access to this field is RO.

Bits [62:61]

Reserved, RES0.

FORCED_NS, bit [60]

When FEAT_MPAMv0p1 is implemented:

In the Secure state, FORCED_NS indicates the state of MPAM3_EL3.FORCE_NS.

`0b0` In the Non-secure state, always reads as 0.
In the Secure state, indicates that MPAM3_EL3.FORCE_NS == 0.

`0b1` In the Secure state, indicates that MPAM3_EL3.FORCE_NS == 1.

Always reads as 0 in the Non-secure state.

Writes are ignored.

Access to this field is RO.

Otherwise:

Reserved, RES0.

Bits [59:55]

Reserved, RES0.

ALTSP_FRCD, bit [54]

When FEAT_RME is implemented and MPAMIDR_EL1.HAS_ALTSP == 1:

Alternative PARTID forced for PARTIDs in this register.

`0b0` The PARTIDs in MPAM1_EL1 and MPAM0_EL1 are using the primary PARTID space.

`0b1` The PARTIDs in MPAM1_EL1 and MPAM0_EL1 are using the alternative PARTID space.

This bit indicates that a higher Exception level has forced the PARTIDs in this register to use the alternative PARTID space defined for the current Security state.

In MPAM1_EL1, it also indicates that MPAM0_EL1 is forced to use alternative PARTID space.

For more information, see *Alternative PARTID space and PARTID virtualization* on page 6-89.

Access to this field is RO.

Otherwise:

Reserved, RES0.

Bits [53:48]

Reserved, RES0.

PMG_D, bits [47:40]

Performance monitoring group property for PARTID_D.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

PMG_I, bits [39:32]

Performance monitoring group property for PARTID_I.

The reset behavior of this field is:

- On a Warm reset, 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.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

PARTID_I, bits [15:0]

Partition ID for instruction accesses made from EL1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing 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 register 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) && MPAM3_EL3.TRAPLOWER == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EL2Enabled() && MPAM2_EL2.TRAPMPAM1EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && 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
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            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) && MPAM3_EL3.TRAPLOWER == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EL2Enabled() && MPAM2_EL2.TRAPMPAM1EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && 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
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            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) && MPAM3_EL3.TRAPLOWER == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return MPAM1_EL1;
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
```

```
    if EL2Enabled() && !ELUsingAArch32(EL2) && 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) && MPAM3_EL3.TRAPLOWER == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            MPAM1_EL1 = X[t];
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && 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 bit [63] is architecturally mapped to AArch64 System register MPAM3_EL3[63] when EL3 is implemented.

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

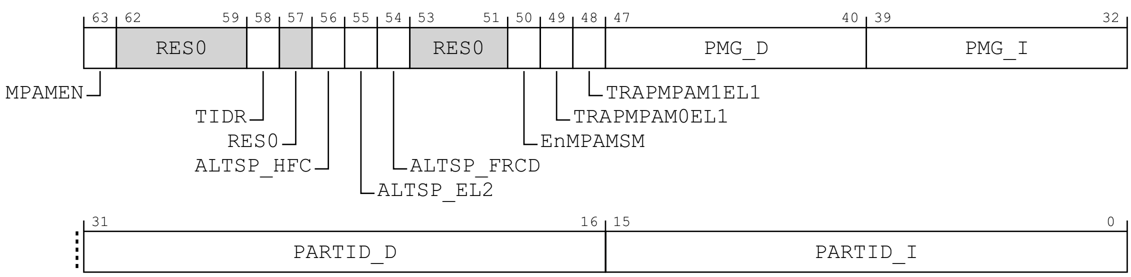
This register is present only when FEAT_MPAM is implemented. Otherwise, direct accesses to MPAM2_EL2 are UNDEFINED.

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

Attributes

MPAM2_EL2 is a 64-bit register.

Field descriptions



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.

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

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

The reset behavior of this field is:

- On a Warm reset, this field resets to `0`.

Accessing this field has the following behavior:

- When EL3 is not implemented, access to this field is RW.
- Otherwise, access to this field is RO.

Bits [62:59]

Reserved, RES0.

TIDR, bit [58]

When (FEAT_MPAMv0p1 is implemented or FEAT_MPAMv1p1 is implemented) and MPAMIDR_EL1.HAS_TIDR == 1:

TIDR traps accesses to MPAMIDR_EL1 from EL1 to EL2.

`0b0` This control does not cause any instructions to be trapped.

`0b1` Trap accesses to MPAMIDR_EL1 from EL1 to EL2.

MPAMHCR_EL2.TRAP_MPAMIDR_EL1 == 1 also traps MPAMIDR_EL1 accesses from EL1 to EL2. If either TIDR or TRAP_MPAMIDR_EL1 are 1, accesses are trapped.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [57]

Reserved, RES0.

ALTSP_HFC, bit [56]

When FEAT_RME is implemented and MPAMIDR_EL1.HAS_ALTSP == 1:

Hierarchical force of alternative PARTID space controls. When MPAM3_EL3.ALTSP_HEN is 0, ALTSP controls in MPAM2_EL2 have no effect. When MPAM3_EL3.ALTSP_HEN is 1, this bit selects whether the PARTIDs in MPAM1_EL1 and MPAM0_EL1 are in the primary (0) or alternative (1) PARTID space for the security state.

`0b0` When MPAM3_EL3.ALTSP_HEN is 1, the PARTID space of MPAM1_EL1.PARTID_I, MPAM1_EL1.PARTID_D, MPAM0_EL1.PARTID_I, and MPAM0_EL1.PARTID_D are in the primary PARTID space for the Security state.

`0b1` When MPAM3_EL3.ALTSP_HEN is 1, the PARTID space of MPAM1_EL1.PARTID_I, MPAM1_EL1.PARTID_D, MPAM0_EL1.PARTID_I, and MPAM0_EL1.PARTID_D are in the alternative PARTID space for the Security state.

This control has no effect when MPAM3_EL3.ALTSP_HEN is 0.

For more information, see *Alternative PARTID space and PARTID virtualization* on page 6-89.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

ALTSP_EL2, bit [55]

When FEAT_RME is implemented and MPAMIDR_EL1.HAS_ALTSP == 1:

Select alternative PARTID space for PARTIDs in MPAM2_EL2 when MPAM3_EL3.ALTSP_HEN is 1.

`0b0` When MPAM3_EL3.ALTSP_HEN is 1, selects the primary PARTID space for MPAM2_EL2.PARTID_I and MPAM2_EL2.PARTID_D.

`0b1` When MPAM3_EL3.ALTSP_HEN is 1, selects the alternative PARTID space for MPAM2_EL2.PARTID_I and MPAM2_EL2.PARTID_D.

For more information see *Alternative PARTID space and PARTID virtualization* on page 6-89.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

ALTSP_FRCD, bit [54]

When FEAT_RME is implemented and MPAMIDR_EL1.HAS_ALTSP == 1:

Alternative PARTID forced for PARTIDs in this register.

`0b0` The PARTIDs in this register are using the primary PARTID space.

`0b1` The PARTIDs in this register are using the alternative PARTID space.

This bit indicates that a higher Exception level has forced the PARTIDs in this register to use the alternative PARTID space defined for the current Security state. In EL2, it is also 1 when MPAM2_EL2.ALTSP_EL2 is 1.

For more information, see *Alternative PARTID space and PARTID virtualization* on page 6-89.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Access to this field is RO.

Otherwise:

Reserved, RES0.

Bits [53:51]

Reserved, RES0.

EnMPAMSM, bit [50]

When FEAT_SME is implemented:

Traps execution at EL1 of instructions that directly access the MPAMSM_EL1 register to EL2. The exception is reported using ESR_ELx.EC value `0x18`.

`0b0` This control causes execution of these instructions at EL1 to be trapped.

`0b1` This control does not cause execution of any instructions to be trapped.

This field has no effect on accesses to MPAMSM_EL1 from EL2 or EL3.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TRAPMPAM0EL1, bit [49]

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.

The reset behavior of this field is:

- On a Warm reset:
  - When EL3 is not implemented, this field resets to `1`
  - When EL3 is implemented, this field resets to an architecturally UNKNOWN value.

TRAPMPAM1EL1, bit [48]

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.

The reset behavior of this field is:

- On a Warm reset:
  - When EL3 is not implemented,this field resets to `1`.
  - When EL3 is implemented, this field resets to an architecturally UNKNOWN value.

PMG_D, bits [47:40]

Performance monitoring group for data accesses.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

PMG_I, bits [39:32]

Performance monitoring group for instruction accesses.

The reset behavior of this field is:

- On a Warm reset, 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.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

PARTID_I, bits [15:0]

Partition ID for instruction accesses made from EL2.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing 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 register 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
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            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
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            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) && MPAM3_EL3.TRAPLOWER == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EL2Enabled() && MPAM2_EL2.TRAPMPAM1EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && 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
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            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) && MPAM3_EL3.TRAPLOWER == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EL2Enabled() && MPAM2_EL2.TRAPMPAM1EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && 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
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            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 bit [63] is architecturally mapped to AArch64 System register MPAM2_EL2[63] when EL2 is implemented.

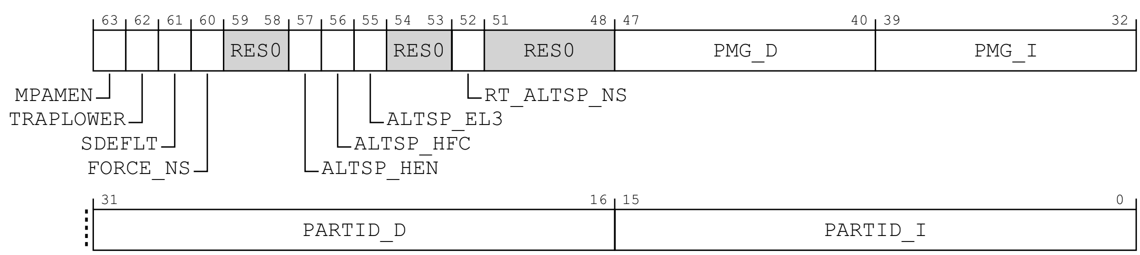
AArch64 System register MPAM3_EL3 bit [63] is architecturally mapped to AArch64 System register MPAM1_EL1[63].

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

Attributes

MPAM3_EL3 is a 64-bit register.

Field descriptions



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.

The reset behavior of this field is:

- On a Warm reset, this field resets to `0`.

Access to this field is RW.

TRAPLOWER, bit [62]

Trap direct accesses to 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 direct accesses of MPAM System registers to trap to EL3.

The reset behavior of this field is:

- On a Warm reset, this field resets to 1.

SDEFLT, bit [61]

When (FEAT_MPAMv0p1 is implemented or FEAT_MPAMv1p1 is implemented) and MPAMIDR_EL1.HAS_SDEFLT == 1:

SDEFLT overrides the PARTID and PMG with the default PARTID and default PMG when executing in the Secure state.

`0b0` The PARTID and PMG are determined normally in the Secure state.

`0b1` When executing in the Secure state, the PARTID is always PARTID 0, and the PMG is always PMG 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

FORCE_NS, bit [60]

When FEAT_MPAMv0p1 is implemented and MPAMIDR_EL1.HAS_FORCE_NS == 1:

FORCE_NS forces MPAM_NS to always be 1 in the Secure state.

`0b0` MPAM_NS is 0 when executing in the Secure state.

`0b1` MPAM_NS is 1 when executing in the Secure state.

An implementation is permitted to have this field as RAO if the implementation does not support generating MPAM_NS as 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [59:58]

Reserved, RES0.

ALTSP_HEN, bit [57]

When FEAT_RME is implemented and MPAMIDR_EL1.HAS_ALTSP == 1:

Hierarchical enable for alternative PARTID space controls. Alternative PARTID space controls in MPAM2_EL2 have no effect when this field is zero.

`0b0` Disable alternative PARTID space controls in MPAM2_EL2. The PARTID space for PARTIDs in MPAM2_EL2, MPAM1_EL1, and MPAM0_EL1 is selected by MPAM3_EL3.ALTSP_HFC.

`0b1` Enable alternative PARTID space controls in MPAM2_EL2 to control the PARTID space used for PARTIDs in MPAM2_EL2, MPAM1_EL1, and MPAM0_EL1.

For more information, see *Alternative PARTID space and PARTID virtualization* on page 6-89.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

ALTSP_HFC, bit [56]

When FEAT_RME is implemented and MPAMIDR_EL1.HAS_ALTSP == 1:

Hierarchical force of alternative PARTID space controls. When MPAM3_EL3.ALTSP_HEN is 0, the PARTID space for PARTIDs in MPAM2_EL2, MPAM1_EL1, and MPAM0_EL1 is selected by the value of this bit.

`0b0` When MPAM3_EL3.ALTSP_HEN is 0, the PARTID space of MPAM2_EL2.PARTID, MPAM1_EL1.PARTID and MPAM0_EL1.PARTID are the primary PARTID space for the security state.

`0b1` When MPAM3_EL3.ALTSP_HEN is 0, the PARTID space of MPAM2_EL2.PARTID and MPAM1_EL1.PARTID and MPAM0_EL1.PARTID are the alternative PARTID space for the security state.

For more information, see *Alternative PARTID space and PARTID virtualization* on page 6-89.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

ALTSP_EL3, bit [55]

When FEAT_RME is implemented and MPAMIDR_EL1.HAS_ALTSP == 1:

Select alternative PARTID space for PARTIDs in MPAM3_EL3.

`0b0` Selects the primary PARTID space of MPAM3_EL3.PARTID_I and MPAM3_EL3.PARTID_D.

`0b1` Selects the alternative PARTID space of MPAM3_EL3.PARTID_I and MPAM3_EL3.PARTID_D.

For more information, see *Alternative PARTID space and PARTID virtualization* on page 6-89.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [54:53]

Reserved, RES0.

RT_ALTSP_NS, bit [52]

When FEAT_RME is implemented and MPAMIDR_EL1.HAS_ALTSP == 1:

Alternative PARTID space selection for the Root security state.

`0b0` The alternative PARTID space in the Root security state is the Secure PARTID space.

`0b1` The alternative PARTID space in the Root security state is the Non-secure PARTID space.

This field has no effect except in the Root security state (EL3).

The reset behavior of this field is:

- On a Warm reset, this field resets to `0`.

Otherwise:

Reserved, RES0.

Bits [51:48]

Reserved, RES0.

PMG_D, bits [47:40]

Performance monitoring group for data accesses.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

PMG_I, bits [39:32]

Performance monitoring group for instruction accesses.

The reset behavior of this field is:

- On a Warm reset, 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 EL3.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

PARTID_I, bits [15:0]

Partition ID for instruction accesses made from EL3.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing 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 register 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

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

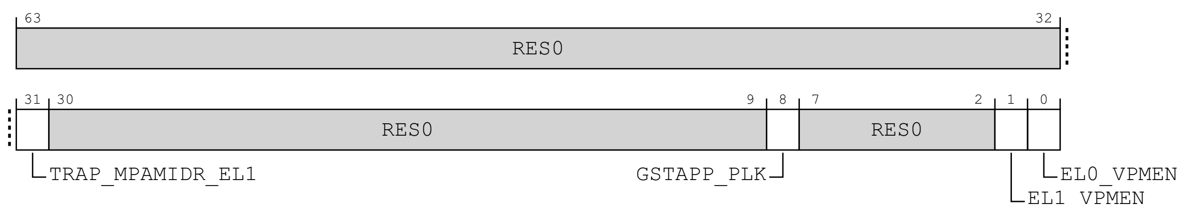
This register is present only when FEAT_MPAM is implemented and MPAMIDR_EL1.HAS_HCR == 1. Otherwise, direct accesses to MPAMHCR_EL2 are UNDEFINED.

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

Attributes

MPAMHCR_EL2 is a 64-bit register.

Field descriptions



Bits [63:32]

Reserved, RES0.

TRAP_MPAMIDR_EL1, bit [31]

Trap accesses from EL1 to MPAMIDR_EL1 to EL2.

`0b0` This control does not cause any instructions to be trapped.

`0b1` Direct accesses to MPAMIDR_EL1 from EL1 are trapped to EL2.

The reset behavior of this field is:

- On a Warm reset:
  - When EL3 is not implemented,this field resets to 1.
  - When EL3 is implemented, this field resets to an architecturally UNKNOWN value.

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.

The reset behavior of this field is:

- On a Warm reset, 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.

The reset behavior of this field is:

- On a Warm reset, 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.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing MPAMHCR_EL2

Accesses to this register use the following encodings in the System register 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
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
```

```
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            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
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            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

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

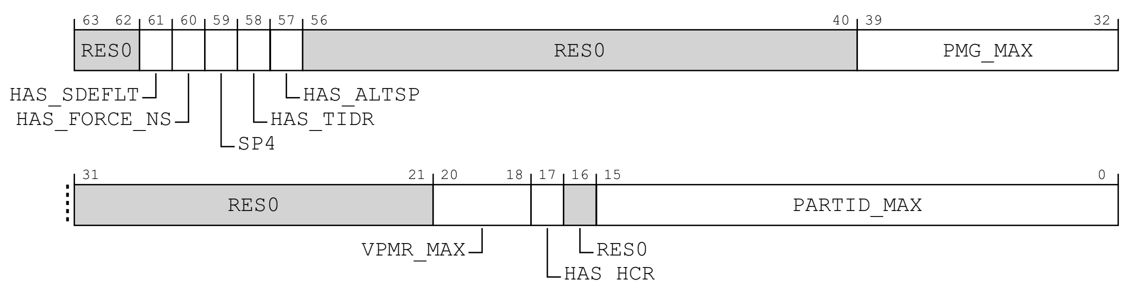
Configurations

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

Attributes

MPAMIDR_EL1 is a 64-bit register.

Field descriptions



MPAMIDR_EL1 indicates the MPAM implementation parameters of the PE.

Bits [63:62]

Reserved, RES0.

HAS_SDEFLT, bit [61]

HAS_SDEFLT indicates support for MPAM3_EL3.SDEFLT bit. Defined values are:

`0b0` The SDEFLT bit is not implemented in MPAM3_EL3.

`0b1` The SDEFLT bit is implemented in MPAM3_EL3.

When MPAM3_EL3.SDEFLT == 1, accesses from the Secure Execution state use the default PARTID, PARTID == 0.

HAS_FORCE_NS, bit [60]

HAS_FORCE_NS indicates support for MPAM3_EL3.FORCE_NS bit. Defined values are:

`0b0` The FORCE_NS bit is not implemented in MPAM3_EL3.

`0b1` The FORCE_NS bit is implemented in MPAM3_EL3.

When MPAM3_EL3.FORCE_NS == 1, accesses from the Secure Execution state have MPAM_NS == 1.

SP4, bit [59]

Supports 4 MPAM PARTID spaces.

`0b0` MPAM supports 2 PARTID spaces.

`0b1` MPAM supports 4 PARTID spaces.

HAS_TIDR, bit [58]

HAS_TIDR indicates support for MPAM2_EL2.TIDR bit. Defined values are:

`0b0` The TIDR bit is not implemented in MPAM2_EL2.

`0b1` The TIDR bit is implemented in MPAM2_EL2.

HAS_ALTSP, bit [57]

HAS_ALTSP indicates support for alternative PARTID spaces.

`0b0` Alternative PARTID spaces are not implemented.

`0b1` Alternative PARTID spaces are implemented with control bits in MPAM3_EL3 and MPAM2_EL2.

Bits [56:40]

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]

When MPAMIDR_EL1.HAS_HCR == 1:

Indicates the maximum register index n for the MPAMVPM<n>_EL2 registers.

Otherwise:

Reserved, RAZ.

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 PARTID that the implementation can generate. The PARTID_I and PARTID_D fields of every MPAMn_ELx must implement at least enough bits to represent PARTID_MAX.

Accessing MPAMIDR_EL1

Accesses to this register use the following encodings in the System register 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) && MPAM3_EL3.TRAPLOWER == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EL2Enabled() && MPAMIDR_EL1.HAS_HCR == '1' && MPAMHCR_EL2.TRAP_MPAMIDR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MPAMIDR_EL1.HAS_TIDR == '1' && MPAM2_EL2.TIDR == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        return MPAMIDR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            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 FEAT_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.

Attributes

MPAMVPM0_EL2 is a 64-bit register.

Field descriptions

63 - 48	47 - 32
PhyPARTID3	PhyPARTID2

31 - 16	15 - 0
PhyPARTID1	PhyPARTID0

PhyPARTID3, bits [63:48]

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

The reset behavior of this field is:

- On a Warm reset, 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.

The reset behavior of this field is:

- On a Warm reset, 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.

The reset behavior of this field is:

- On a Warm reset, 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.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing MPAMVPM0_EL2

Accesses to this register use the following encodings in the System register 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
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            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
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
```

```
    if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            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 FEAT_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.

Attributes

MPAMVPM1_EL2 is a 64-bit register.

Field descriptions

63 – 48	47 – 32
PhyPARTID7	PhyPARTID6

31 – 16	15 – 0
PhyPARTID5	PhyPARTID4

PhyPARTID7, bits [63:48]

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

The reset behavior of this field is:

- On a Warm reset, 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.

The reset behavior of this field is:

- On a Warm reset, 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.

The reset behavior of this field is:

- On a Warm reset, 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.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing MPAMVPM1_EL2

Accesses to this register use the following encodings in the System register 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
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            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
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
```

```
    if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            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 FEAT_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.

Attributes

MPAMVPM2_EL2 is a 64-bit register.

Field descriptions

63 - 48	47 - 32
PhyPARTID11	PhyPARTID10

31 - 16	15 - 0
PhyPARTID9	PhyPARTID8

PhyPARTID11, bits [63:48]

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

The reset behavior of this field is:

- On a Warm reset, 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.

The reset behavior of this field is:

- On a Warm reset, 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.

The reset behavior of this field is:

- On a Warm reset, 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.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing MPAMVPM2_EL2

Accesses to this register use the following encodings in the System register 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
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            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
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
```

```
        if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                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 FEAT_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.

Attributes

MPAMVPM3_EL2 is a 64-bit register.

Field descriptions

63 - 48	47 - 32
PhyPARTID15	PhyPARTID14

31 - 16	15 - 0
PhyPARTID13	PhyPARTID12

PhyPARTID15, bits [63:48]

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

The reset behavior of this field is:

- On a Warm reset, 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.

The reset behavior of this field is:

- On a Warm reset, 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.

The reset behavior of this field is:

- On a Warm reset, 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.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing MPAMVPM3_EL2

Accesses to this register use the following encodings in the System register 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
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            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
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
```

```
    if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            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 FEAT_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.

Attributes

MPAMVPM4_EL2 is a 64-bit register.

Field descriptions

63–48	47–32
PhyPARTID19	PhyPARTID18

31–16	15–0
PhyPARTID17	PhyPARTID16

PhyPARTID19, bits [63:48]

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

The reset behavior of this field is:

- On a Warm reset, 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.

The reset behavior of this field is:

- On a Warm reset, 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.

The reset behavior of this field is:

- On a Warm reset, 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.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing MPAMVPM4_EL2

Accesses to this register use the following encodings in the System register 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
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            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
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
```

```
    if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            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 FEAT_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.

Attributes

MPAMVPM5_EL2 is a 64-bit register.

Field descriptions

63–48	47–32
PhyPARTID23	PhyPARTID22

31–16	15–0
PhyPARTID21	PhyPARTID20

PhyPARTID23, bits [63:48]

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

The reset behavior of this field is:

- On a Warm reset, 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.

The reset behavior of this field is:

- On a Warm reset, 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.

The reset behavior of this field is:

- On a Warm reset, 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.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing MPAMVPM5_EL2

Accesses to this register use the following encodings in the System register 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
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            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
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
```

```
    if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            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 FEAT_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.

Attributes

MPAMVPM6_EL2 is a 64-bit register.

Field descriptions

63 - 48	47 - 32
PhyPARTID27	PhyPARTID26

31 - 16	15 - 0
PhyPARTID25	PhyPARTID24

PhyPARTID27, bits [63:48]

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

The reset behavior of this field is:

- On a Warm reset, 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.

The reset behavior of this field is:

- On a Warm reset, 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.

The reset behavior of this field is:

- On a Warm reset, 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.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing MPAMVPM6_EL2

Accesses to this register use the following encodings in the System register 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
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            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
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
```

```
    if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            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

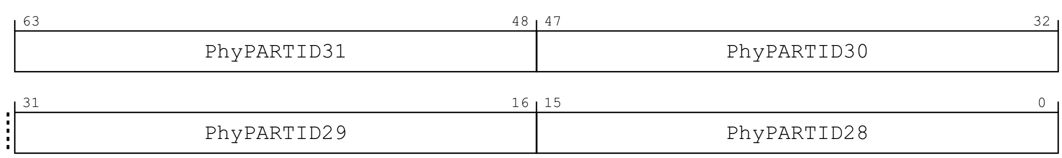
This register is present only when FEAT_MPAM is implemented, MPAMIDR_EL1.HAS_HCR == 1 and MPAMIDR_EL1.VPMR_MAX == 7. Otherwise, direct accesses to MPAMVPM7_EL2 are UNDEFINED.

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

Attributes

MPAMVPM7_EL2 is a 64-bit register.

Field descriptions



PhyPARTID31, bits [63:48]

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

The reset behavior of this field is:

- On a Warm reset, 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.

The reset behavior of this field is:

- On a Warm reset, 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.

The reset behavior of this field is:

- On a Warm reset, 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.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing MPAMVPM7_EL2

Accesses to this register use the following encodings in the System register 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
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            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
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
```

```
    if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            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

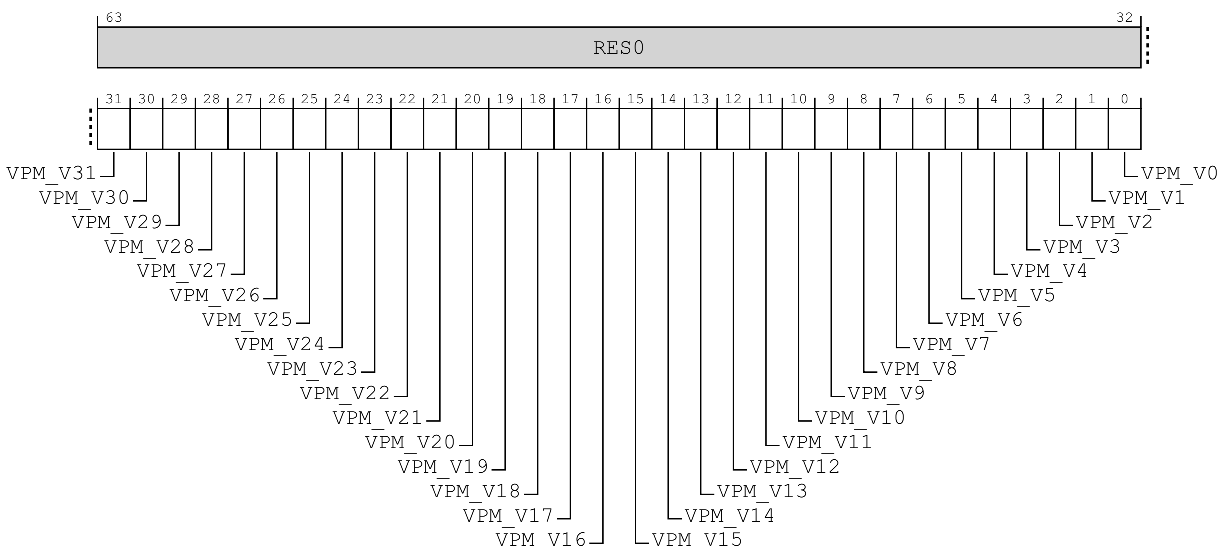
This register is present only when FEAT_MPAM is implemented and MPAMIDR_EL1.HAS_HCR == 1. Otherwise, direct accesses to MPAMVPMV_EL2 are UNDEFINED.

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

Attributes

MPAMVPMV_EL2 is a 64-bit register.

Field descriptions



Bits [63:32]

Reserved, RES0.

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

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

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing MPAMVPMV_EL2

Accesses to this register use the following encodings in the System register 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
        if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        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
        if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        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 Exception level 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 Exception level.

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 3-37 for more on default IDs.

The MPAMEN bit is reset to 0.

7.6 SDEFLT

In MPAM v0.1 and from MPAM v1.1, when MPAMIDR_EL1.HAS_SDEFLT is 1, the writeable MPAM3_EL3.SDEFLT is implemented. When MPAMIDR_EL1.HAS_SDEFLT is 0, MPAM3_EL3.SDEFLT is RES0, and Secure PARTID generation is as if no SDEFLT functionality is present.

The SDEFLT bit only affects the generation of MPAM PARTIDs from the Secure state. When MPAM3_EL3.SDEFLT is 1:

- A Secure physical PARTID is always generated as the default Secure PARTID. If MPAMIDR_EL1.HAS_FORCE_NS is 1 and MPAM3_EL3.FORCE_NS is 1, the generated PARTID is the default Non-secure PARTID, otherwise it is the default Secure PARTID.
- A PMG is always generated as the default PMG.

When the SDEFLT bit is 0, all accesses have the PARTID and PMG generated as normal. For more information on default IDs, see *PARTID spaces and properties* on page 3-37.

The SDEFLT bit is reset to an UNKNOWN value.

7.6.1 Interaction of SDEFLT and MPAMEN

In MPAM v0.1 and from MPAM v1.1, when MPAMIDR_EL1.HAS_SDEFLT is 1, the writeable MPAM3_EL3.SDEFLT is implemented:

- When MPAMEN is 0, all accesses from Secure and Non-secure states have the physical PARTID and the PMG generated as 0.
- When MPAMEN is 1 and SDEFLT is 0, all accesses from Secure and Non-secure states have the PARTID and PMG generated as normal.
- When MPAMEN is 1 and SDEFLT is 1:
  — All accesses from Secure state have the physical PARTID as 0 and PMG as 0.
  — All accesses from Non-secure state have the PARTID and PMG generated as normal.

7.7 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. HCR_EL2.NV == 1 does not affect accesses from EL0, EL2, or EL3. When HCR_EL2.NV == 1 and MPAM3_EL3.TRAPLOWER == 1, accesses to MPAM registers from EL2 are trapped to EL3. See *Nested virtualization extension* on page 6-82 for details.

HCR_EL2.NV2 == 1 alters the behavior of MPAM3_EL3.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 6-82 for details.

7.8 FORCE_NS

In MPAMv0.1, when MPAMIDR_EL1.HAS_FORCE_NS is 1, MPAM3_EL3.FORCE_NS is implemented, and must implement either one of two behaviors:

- Secure PARTIDs are not implemented.
- FORCE_NS can be written by software.

If an implementation does not implement Secure PARTIDs, then MPAM3_EL3.FORCE_NS is RAO/WI and MPAM_NS is always generated as 1 on accesses from Secure state and Non-secure state.

If an implementation allows MPAM3_EL3.FORCE_NS to be written by software, then:

- MPAM3_EL3.FORCE_NS is reset to 0.
- When MPAM3_EL3.FORCE_NS is 0, MPAM_NS is generated as 0 on accesses from Secure state.
- When MPAM3_EL3.FORCE_NS is 1, MPAM_NS is generated as 1 on accesses from Secure state.
- Generation of MPAM_NS from Non-secure state is unaffected.

When MPAMIDR_EL1.HAS_FORCE_NS is 0, MPAM3_EL3.FORCE_NS is RES0, and Secure PARTID generation is as if no FORCE_NS functionality is present.

The FORCE_NS bit is reset to an UNKNOWN value.

7.9 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.
- If MPAMIDR_EL1.HAS_FORCE_NS is 1 and MPAM3_EL3.FORCE_NS is implemented as writeable, MPAM3_EL3.FORCE_NS must be reset to 0 on a Warm or Cold reset of the PE.
- The power and reset domain of each MSC component is specific to that component.

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.

7.10 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 FEAT_SEL2 is implemented, Secure EL2 does not exist in the Armv8 Architecture before v8.4. FEAT_SEL2 is permitted to be implemented or not implemented in v8.4 or later implementations. If FEAT_SEL2 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.

7.10.1 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.

7.10.2 Effects if EL2 is implemented in neither Security state

- MPAM2_EL2 is RES0 when accessed from EL3. It is UNDEFINED from all other Exception levels.
- 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 Exception level.
- 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.

7.10.3 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 Exception levels.
- 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 Exception levels.
- MPAMVPM0_EL2 through MPAMVPM7_EL2 are RW when accessed from EL3 or Non-secure EL2. These registers are UNDEFINED from all other Exception levels.

If an implementation supports Secure state and Secure EL2 does not exist, all behaviors listed in *Effects if EL2 is implemented only in Non-secure state, or if implemented but disabled by SCR_EL2.EEL2 = 0 in Secure state* on page 7-152 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 in *Effects if EL2 is implemented only in Non-secure state, or if implemented but disabled by SCR_EL2.EEL2 = 0 in Secure state* on page 7-152 must be followed by the MPAM implementation on the Secure side.

If Non-secure EL2 exists, the behaviors in *Effects if EL2 is implemented in neither Security state* on page 7-152 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, or 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 Requester. These include cache memories, interconnects, memory management units, memory channel controllers, queues, buffers, and rate adaptors.

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.

8.1.1 MPAM versions in MSCs

MSCs can be used in MPAM v1.0, v1.1, and in v0.1 under certain conditions. If an MSC does not implement any of the MPAM v1.1 MSC features listed in *MPAM versions for PEs* on page 1-22, it is version 1.0.

Note

The MPAM version of an MSC is available in MPAMF_AIDR, see *MPAM versions for MSCs* on page 1-22.

If an MSC implements the extended MPAMF_IDR, or any of the MPAM v1.1 MSC features, it is either MPAM v1.1 or MPAM v0.1. An MSC must not use MPAM v0.1 unless all of these conditions are met:

- The MSC can initiate requests.
- Requests can be initiated in the Secure address space.
- Requests to the Secure address space can have MPAM_NS forced to 1.
- Software that configures the MSC to make requests in the Secure address space:
  - Cannot control the forcing of MPAM_NS.
  - Cannot easily see that MPAM_NS is being forced.

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.

For more information, see:

- *Model of a resource partitioning control* on page 5-54 for a model of a resource partitioning control.
- Chapter 9 *Resource Partitioning Controls* for more detailed information on resource partitioning controls.
- *Resource instance selection* on page 8-158 for how these controls are affected when resource instance selection is supported.

8.3 Resource instance selection

Resource instance selection, or RIS, allows support for MSCs with multiple resources, including multiple resources with the same type or partitioning control. This means that each MSC can only have independent resource controls and two or more resources of the same type when RIS is implemented. In MPAM v0.1 and from MPAM v1.1, this optional feature is implemented when MPAMF_IDR.HAS_RIS is 1.

This section provides more details on:

- *RIS values* on page 8-158.
- *RIS controls in MPAMCFG_PART_SEL* on page 8-158.
- *RIS controls in MSMON_CFG_MON_SEL* on page 8-160.
- *Effects of MPAMCFG_PART_SEL.RIS on values read from other registers* on page 8-159.
- *Selecting a resource to monitor* on page 8-160.
- *Undefined RIS values* on page 8-160.
- *Reporting errors involving RIS* on page 8-161.

8.3.1 RIS values

Each partitionable or monitorable resource is associated with a unique RIS value.

MPAM resource monitors are usually associated with a resource instance, and the RIS value for that resource instance is also used in MSMON_CFG_MON_SEL.RIS to select the monitors associated with that resource.

Information on what RIS value is assigned to which resource instance of the MSC is not available from MSC IDRs, and must be provided by means other than the hardware ID registers.

MPAMF_IDR.RIS_MAX gives the largest value of RIS that is defined for the MSC. A RIS value from 0 to RIS_MAX can be assigned to any partitioned or monitored resource. There is no requirement for every RIS value to be assigned to a partitioned or monitored resource.

As software for MPAMv1.0 would not set the value of the RIS field to any value other than 0, the only resource that can be identified and controlled by software that is not aware of this feature is resource instance 0.

8.3.2 RIS controls in MPAMCFG_PART_SEL

The value in MPAMCFG_PART_SEL.RIS selects the resource instance that is:

- Described by the MPAMF ID registers.
- Controlled by accessing the MPAMCFG_* registers.

8.3.3 Effects of MPAMCFG_PART_SEL.RIS on partitioning controls

To access control settings for a particular resource instance and PARTID, MPAMCFG_PART_SEL.PART_SEL is set to the PARTID and MPAMCFG_PART_SEL.RIS is set to the value associated with that resource instance. Accesses to additional MPAMCFG_* registers made without changing MPAMCFG_PART_SEL can be used to read and write additional control settings for that resource instance and partition.

If a control applies to all resource instances, this common control must be accessed with MPAMCFG_PART_SEL.RIS set to 0.

If there is only a single resource instance in an MSC, all controls must be associated with MPAMCFG_PART_SEL.RIS set to 0.

If an MPAMCFG register is accessed when MPAMCFG_PART_SEL.RIS is set to a resource instance that does not support the accessed control, then the behavior is CONSTRAINED UNPREDICTABLE, see *RIS in MPAMCFG_PART_SEL.RIS does not have partitioning control (errorcode == 9)* on page 12-367.

8.3.4 Effects of MPAMCFG_PART_SEL.RIS on values read from other registers

Fields within other registers reflect the capabilities of the resource instance that has been selected by MPAMCFG_PART_SEL.RIS, and so might have different values in different resource instances, as in Table 8-1 on page 8-159.

Table 8-1 Fields affected by a resource instance

Register	Field	Description for resource instance
MPAMF_CCAP_IDR	CMAX_WD	This field is permitted to vary between resource instances.
MPAMF_CPOR_IDR	CPBM_WD	This field is permitted to vary between resource instances.
MPAMF_CSUMON_IDR	HAS_CAPTURE	This field is permitted to vary between resource instances.
	CSU_RO	This field is permitted to vary between resource instances.
	NUM_MON	This field is permitted to vary between resource instances.
MPAMF_IDR	NO_IMPL_MSMON	MPAMF_IMPL_IDR describes no resource usage monitors.
	NO_IMPL_PART	MPAMF_IMPL_IDR describes no resource partitioning controls.
	HAS_MSMON	The resource usage monitors described in MPAMF_MSMON_IDR, otherwise this field is `0b0`.
	HAS_IMPL_IDR	The IMPLEMENTATION DEFINED features described in MPAMF_IMPL_IDR, otherwise this field is `0b0`.
	HAS_PRI_PART	The priority partitioning described in MPAMF_PRI_IDR, otherwise `0b0`.
	HAS_MBW_PART	The memory bandwidth partitioning described in MPAMF_MBW_IDR, otherwise `0b0`.
	HAS_CPOR_PART	The cache portion partitioning described in MPAMF_CPOR_IDR, otherwise `0b0`.
	HAS_CCAP_PART	The cache capacity partitioning described in MPAMF_CCAP_IDR, otherwise `0b0`.
MPAMF_IMPL_IDR	IMPLEMENTATION DEFINED	The IMPLEMENTATION DEFINED contents of this register vary according to the resource instance selected, and cannot be specified by the architecture.
MPAMF_MSMON_IDR	MSMON_MBWU	The memory bandwidth usage monitors of the resource, otherwise this field is `0b0`.
	MSMON_CSU	The cache storage usage monitors of the selected resource instance. Otherwise this field is `0b0`.
MPAMF_PRI_IDR[a]	DSPRI_WD	The downstream priority width. Ignored if MPAMF_PRI_IDR.HAS_DSPRI is set to 0.
	DSPRI_0_IS_LOW	The downstream priority encoded with 0 being the low priority. Ignored if MPAMF_PRI_IDR.HAS_DSPRI is set to 0.
	HAS_DSPRI	The downstream priority control.
	INTPRI_WD	The internal priority width. Ignored if MPAMF_PRI_IDR.HAS_INTPRI is set to 0.
	INTPRI_0_IS_LOW	The internal priority encoded with 0 being low priority. Ignored if MPAMF_PRI_IDR.HAS_INTPRI is set to 0.
	HAS_INTPRI	The internal priority control.
MPAMF_MBW_IDR	All fields	These fields are permitted to vary between resource instances.

Table 8-1 Fields affected by a resource instance (continued)

Register	Field	Description for resource instance
MPAMF_MBWUMON_IDR	HAS_CAPTURE	This field is permitted to vary between resource instances.
	HAS_RWBW	This field is permitted to vary between resource instances.
	HAS_LONG	This field is permitted to vary between resource instances.
	LWD	This field is permitted to vary between resource instances.
	SCALE	This field is permitted to vary between resource instances.
	NUM_MON	This field is permitted to vary between resource instances.

a. If the priority partitioning is local to the resource instance, then all fields might vary between resource instances. If the priority partitioning operates at the MSC level, then MPAMF_PRI_IDR should be non-zero only for when RIS is 0.

MPAMF_AIDR, MPAMF_ECR, MPAMF_ESR, MPAMF_IIDR, MPAMF_PARTID_NRW_IDR, MPAMF_SIDR, and MPAMCFG_PART_SEL are not affected by RIS.

8.3.5 RIS controls in MSMON_CFG_MON_SEL

The value in MSMON_CFG_MON_SEL.RIS selects the resource instance that is accessed by:

- The MSMON_CFG_* monitor configuration registers.
- The MSMON_* monitor and monitor capture registers.

To access the configuration, value and capture registers associated with a monitor, the value of MSMON_CFG_MON_SEL.RIS should be set to match the RIS value associated with that monitor. Monitors not associated with any particular resource or associated with the MSC must be associated with MPAMCFG_PART_SEL.RIS == 0.

Note

Monitoring ID registers, MPAMF_MSMON_IDR, MPAMF_MBWUMON_IDR, and MPAMF_CSUMON_IDR, are not affected by MSMON_CFG_MON_SEL.RIS. These registers are affected by MPAMCFG_PART_SEL.RIS.

8.3.6 Selecting a resource to monitor

To select the monitors for a particular resource instance, the value of MSMON_CFG_MON_SEL.RIS must be the same value as used in MPAMCFG_PART_SEL.RIS. Monitors that are not associated with an MPAM partitioned resource instance must be selected with a RIS value of 0.

To access a monitor for a particular resource, the MSMON_CFG_MON_SEL.RIS must be set to the resource instance. Then one or more MSMON_CFG_* registers for the particular monitor are accessed.

Any access to a MSMON_* register address will access the register associated with the resource instance value held in MSMON_CFG_MON_SEL.RIS. The exceptions to this are accesses to the MSMON_CFG_MON_SEL and MSMON_CAPT_EVNT registers, which are not affected by the value held in MSMON_CFG_MON_SEL.RIS.

8.3.7 Undefined RIS values

This section covers behaviors when the value of MPAMCFG_PART_SEL.RIS or MSMON_CFG_MON_SEL.RIS:

- Is greater than MPAMF_IDR.RIS_MAX.
- Does not correspond to an MPAM resource implemented in this MSC.
- Does correspond to an implemented MPAM resource, but the selected resource does not implement the control or monitor that has been accessed.

An MSC is permitted to:

- Implement fewer RIS bits than the architecture defines, though it must implement at least enough bits to represent MPAMF_IDR.RIS_MAX.

- Leave some RIS values that are within the range of 0 to MPAMF_IDR.RIS_MAX as undefined.
- Use only the implemented bits to decode RIS for selecting a resource instance.

Undefined resources that are within the range can still be identified. This is because the HAS_* fields within the ID registers all read as 0 when MPAMCFG_PART_SEL.RIS selects an undefined resource. All RIS values greater than MPAMF_IDR.RIS_MAX are undefined.

If software honors MPAMF_IDR.RIS_MAX and avoids accessing any MMR that are not indicated with the corresponding HAS_* fields in the ID registers for that resource instance, it will not cause any RIS-related errors.

For more information on behavior caused by undefined RIS values, see:

- *Undefined RIS in MPAMCFG_PART_SEL.RIS (errorcode == 8)* on page 12-367.
- *RIS in MPAMCFG_PART_SEL.RIS does not have partitioning control (errorcode == 9)* on page 12-367.
- *Undefined RIS in MSMON_CFG_MON_SEL.RIS (errorcode == 10)* on page 12-367.
- *RIS selected by MSMON_CFG_MON_SEL.RIS does not have monitor type (errorcode == 11)* on page 12-368.

Reading an MPAMF ID register when MPAMCFG_PART_SEL is an undefined RIS value

Access to an MPAMF ID register when MPAMCFG_PART_SEL.RIS is an undefined value must produce an ID register value where all HAS_* fields read as 0. This action does not produce an error in MPAMF_ESR or signal an error interrupt.

8.3.8 Reporting errors involving RIS

Software could misconfigure the RIS fields in MPAMCFG_PART_SEL and MSMON_CFG_MON_SEL registers, possibly resulting in errors. See *Optionality of error detection and reporting* on page 12-375.

When an error is reported that involves a RIS value, the MPAMF_ESR.RIS field must be set to:

- For errors involving MPAMCFG_* register accesses, the MPAMCFG_PART_SEL.RIS
- For errors involving MSMON_* register accesses, the MSMON_CFG_MON_SEL.RIS value.

For MPAM errors that do not capture the RIS field in MPAMF_ESR.RIS as shown in Table 12-1 on page 12-365, MPAMF_ESR.RIS should be set to 0.

8.4 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 3-37.
- 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* on page 8-162.
- 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).

8.4.1 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.

8.4.2 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.

8.5 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.

8.5.1 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 the memory-mapped configuration registers of an MSC should 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. For more information on recommended configurations of memory-mapped registers of an MSC, see *MPAM feature page* on page 11-203.

8.6 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* .

8.7 System-wide PARTID and PMG widths

This section is informative.

The behavior of an MSC is CONSTRAINED UNPREDICTABLE if it receives an MPAM PARTID or PMG outside the range it supports. For more information, see *Behavior of configuration reads and writes with errors* on page 12-370.

For predictable behavior, the PARTID on a request by a Requester 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 Requester 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 7-95, and *Determining presence and location of MMRs* on page 11-202.

8.8 MPAM interrupts

This section is *normative*.

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

- MPAM Error Interrupt.
- MPAM Overflow Interrupt.

8.8.1 MPAM Error Interrupt

MPAM errors in MSCs are described in *Error conditions in accessing memory-mapped registers* on page 12-365.

MPAM errors that are detected in an MSC are recorded in MPAMF_ESR and signaled 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 12-365, both the MPAMF_ESR and MPAMF_ECR must be RAZ/WI. An error cannot be encountered if the MSC:

- Does not support any feature of MPAM that can raise that error.
- Is designed so that the error cannot occur.
- Is permitted to have no detection for that error and does not implement detection for the error, see *Required error condition detection* on page 12-375.

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, GIC architecture version 3.0 and version 4.0* 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.

If the MSC supports signaling the MPAM error interrupt through an MSI, the interrupt must be edge-triggered.

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.

If the MSC supports signaling the MPAM error interrupt through an MSI, the interrupt must be edge-triggered.

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

Support for MSI writes to signal error interrupts

Message signaled interrupts (MSIs) are signaled using a memory write that is usually directed at an interrupt translation service.

The support for error MSIs is identified by the MPAMF_IDR.{HAS_ERR_MSI, HAS_ESR} fields.

The registers that contain the error MSI write configuration are:

- MPAMF_ERR_MSI_ADDR_L.
- MPAMF_ERR_MSI_ADDR_H.
- MPAMF_ERR_MSI_ATTR.
- MPAMF_ERR_MSI_DATA.
- MPAMF_ERR_MSI_MPAM.

Instances of these MSI configuration registers exist in each of the Secure physical address space and the Non-secure physical address space. The set of these registers in an address space configures the error MSI write for errors from the MPAMCFG_* or MPAMF_* registers in that address space.

Errors can also be raised by errors in requests. Errors in requests which have the PARTID space selected by MPAM_NS of 0 are signaled as Secure errors using the MSI write information from the MPAMF_ERR_MSI_* registers in the Secure address space. Errors in requests which have the PARTID space selected by MPAM_NS of 1 are signaled as Non-secure errors using the MSI write information from the MPAM_ERR_MSI_* registers in the Non-secure space.

8.8.2 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 10-197).

Support of an overflow interrupt is optional in an MSC. If the MSC has monitors that can overflow, Arm recommends that the MPAM overflow interrupt be implemented.

When an MPAM monitor instance overflows, it sets the OFLOW_STATUS flag in the monitor instance's control register. If the OFLOW_STATUS flag was previously 0 and OFLOW_INTR bit is 1, an overflow interrupt is signaled if the MSC implements overflow interrupts.

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, GIC architecture version 3.0 and version 4.0* 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 interrupt is reset by writing 0 to the OFLOW_STATUS field of all overflowed monitor instances MSMON_CFG_*_CTL register.

If the MSC supports signaling monitor overflow interrupts through an MSI, the MPAM monitor overflow interrupt must be edge-triggered.

Support for MSI writes to signal overflow interrupts

MSIs are signaled using a memory write that is usually directed at an interrupt translation service.

The support for the monitor overflow interrupt is identified by the MPAMF_MSMON_IDR.{HAS_OFLW_INTR, HAS_OFLW_MSI} fields.

The registers that contain the error MSI write configuration are:
- MSMON_OFLOW_MSI_ADDR_L
- MSMON_OFLOW_MSI_ADDR_H.
- MSMON_OFLOW_MSI_ATTR.
- MSMON_OFLOW_MSI_DATA.
- MSMON_OFLOW_MSI_MPAM.

Instances of these MSI configuration registers exist in each of the Secure physical address space and the Non-secure physical address space. The set of these registers in an address space configures the overflow MSI write from overflow events of monitors accessible in that address space.

Monitor overflow status register

The optional MSMON_OFLOW_SR register gives a summary of the overflow status flags (OFLOW_STATUS and OFLOW_STATUS_L) for each RIS and for each monitor type.

This register contains a flag bit per RIS value. Each flag is 0 if all of the OFLOW_STATUS and OFLOW_STATUS_L bits of all monitor types and all instances of each type for the resource instance are 0. Each flag is 1 if any of the overflow status bits for any monitor instance of any type for the resource instance are 1

The register also contains a flag bit for each monitor type. A monitor type flag is 1 if any monitor instance of the type for the resource instance has the OFLOW_STATUS or OFLOW_STATUS_L bit as 1.

MSMON_OFLOW_SR is read-only. The flags are reset when the OFLOW_STATUS and OFLOW_STATUS_L bits monitored by that flag have all be reset to zero.

The presence of MSMON_OFLOW_SR is indicated by MPAMF_MSMON_IDR.HAS_OFLOW_SR == 1.

Monitor type overflow status bitmap registers

In an implementation that has many monitor instances of a monitor type, the number of monitor instances to scan for overflows is large even after consulting MSMON_OFLOW_SR to eliminate most of the RIS and monitor types. To probe one monitor instance requires that the monitor overflow interrupt service routine set MSMON_CFG_MON_SEL to a monitor instance, read MSMON_CFG_<type>_CTL and check one or two bits in value of that register to see if the OFLOW_STATUS or OFLOW_STATUS_L bit is set.

To assist the scanning of many monitor instances, optional overflow status bitmap registers for a monitor type are available for implementation. These overflow status bitmaps can greatly accelerate the scanning.

Each MPAM monitor type can have an optional overflow status register that shows the overflow status flags in a bitmap of 32 monitor instances. The monitor instances shown are selected in MSMON_CFG_MON_SEL where the RIS field selects the resource instance and the MON_SEL field AND 0xFFE0 selects the lowest of the contiguous 32 monitor instances reported in the bitmap.

For the CSU monitor type, the CSU overflow status register is MSMON_CSU_OFSR. The presence of this register is discoverable in MPAMF_CSUMON_IDR.HAS_OFSR.

Note

In most implementations, CSU monitor instances will not be able to overflow as the maximum value in MSMON_CSU is known at design time and will fit within the architectural maximum of MSMON_CSU. In such an implementation, there will be no CSU monitor instance overflows and MSMON_CSU_OFSR has no value.

For the MBWU monitor type, the MBWU overflow status register is MSMON_MBWU_OFSR. The presence of this register is discoverable in MPAMF_MBWUMON_IDR.HAS_OFSR.

8.9 MSC support of MPAM for RME

An RME system supports 4 physical address spaces. MPAM for RME supports the 4 address spaces and 4 PARTID spaces. The MPAM system environment of an RME system is described in *The MPAM for RME system* on page 5-59.

- An MSC that supports the 4 physical address spaces and 4 PARTID spaces is defined as a *four space MPAM MSC*.
- An MSC that supports only 2 physical address spaces and 2 PARTID spaces is defined as a *two space MPAM MSC*.
- Non-MPAM components support either 1, 2 or 4 address spaces but do not support MPAM at all. Non-MPAM devices have no regulated resources and must not have MPAM devices downstream. See *Non-MPAM components* on page 5-64.
- Other combinations of physical address space support and PARTID space support are not permitted.

4 PARTID spaces must be supported in the levels of interconnect that connect RME PEs, but some MSCs might support MPAM with support for only 2 PARTID spaces. See *MPAM for RME propagation of MPAM_SP with requests* on page 4-46.

The MPAM PARTID space in a request and the physical address space accessed by the request are independent in the request. The associations of physical address space and PARTID space are part of the request generation process at a Requester. An MSC must not assume any association between the PARTID space of a request and the physical address space of the request.

8.9.1 Four-space MSC

An MSC that fully supports RME and MPAM must have 4 PARTID spaces and 4 physical address spaces.

In an MSC that supports 4 PARTID spaces and 4 physical address spaces, the MPAMF_IDR.SP4 bit must be 1 when read from any address space and, if RIS is supported, with any MPAMCFG_PART_SEL.RIS value.

MPAMF_BASE_s, MPAMF_BASE_ns, MPAMF_BASE_rt, MPAMF_BASE_rl must all be defined in the firmware table description of the MSC.

The MPAM memory-mapped registers in each address space are at the offsets from the MPAM Feature Page Base address in that address space. Table 8-2 on page 8-170 shows the relationship of address space, the MPAM feature page base address symbol and the contents of that MPAM feature page.

Table 8-2 Relationship of address space, MPAM feature page base address symbol and a description of the contents of that MPAM feature page

Address Space	MPAM Feature Page Base	Description
Non-Secure	MPAMF_BASE_ns	MPAM MSC registers in the Non-secure address space describe and access controls and monitors for Non-secure PARTID space.
Secure	MPAMF_BASE_s	MPAM MSC registers in the Secure address space describe and access controls and monitors for Secure PARTIDs.
Realm	MPAMF_BASE_rl	MPAM MSC registers in the Realm address space describe and access controls and monitors for the Realm PARTID space
Root	MPAMF_BASE_rt	MPAM MSC registers in the Root address space describe and access controls and monitors for the Root PARTID space

The offsets of MPAM memory-mapped registers from the MPAM Feature Page base address are the same for each MPAM Feature page and in each address space. See Table 11-1 on page 11-208 for all MPAM MSC registers. Added fields and accessors for the two physical address spaces for RME are described in this chapter. See Chapter 11 *Memory-Mapped Registers* for Memory-mapped registers from the MPAMF_BASE_* for that address space.

See *Minimum required MPAM memory-mapped registers* on page 11-205 for the required minimum set of MPAM registers accessible from the MPAM Feature Page in any address space. In each address space the MPAM features of the MSC in that address space are described by decoding the fields in MPAMF_IDR. This indicates that additional ID registers are present and further describe the features. MPAM has no requirement that the resource controls and monitors in one address space are the same as those described in another address space.

Instances of the MPAMCFG_* registers must exist in each of the 4 address spaces where MPAMF_*IDR.HAS_* is 1 for a feature that uses those registers.

There must be an instance of MPAMCFG_PART_SEL in each of the 4 address spaces unless there are no resource controls or resource instances in the PARTID space whose control registers are accessed through that physical address space.

Instances of the MSMON_* registers must exist in each address space where the ID registers indicate that the monitor exists.

There must be an instance of MSMON_CFG_MON_SEL in each of the 4 address spaces that contain any monitor registers.

MPAMF_ESR and MPAMF_ECR must exist in each address space in each of the 4 address spaces where MPAMF_IDR.HAS_ESR is 1.

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.

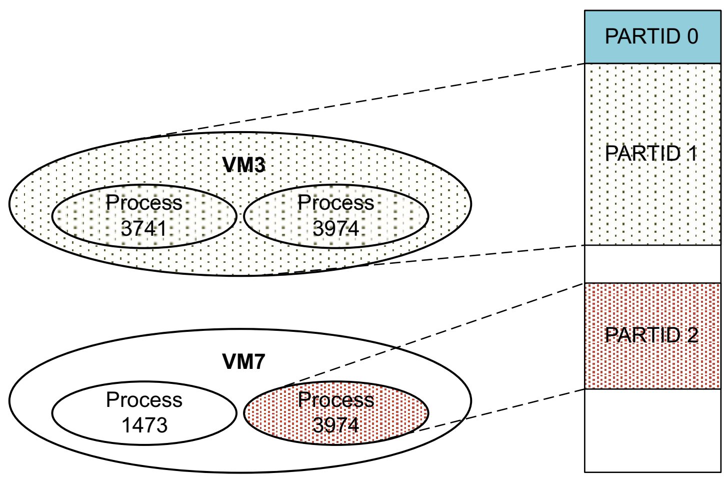


Figure 9-1 Partitioning, VMs, and OS processes

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.

If the implementation supports the RIS MPAM feature, the MSC may have two or more partitionable resources differentiated by the value of MPAMCFG_PART_SEL.RIS. For more information see *Resource instance selection* on page 8-158.

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-391. Cache maximum-capacity partitioning follows the generic maximum-usage control interface described in *Maximum-usage resource controls* on page A-392.

The presence of each standard control is indicated by a bit in MPAMF_IDR, or in a resource-specific memory-mapped ID register. See *Memory-mapped ID register description* on page 11-211.

The control settings storage is accessed through the combination of several access indices:

- The address space used to access the Secure or Non-secure MSC register. Controls for PARTIDs in:
  - The Secure PARTID space are accessed through registers in the Secure address space
  - The Non-secure PARTID space are accessed through registers in the Non-secure address space.
- The MSC that contains the control. This is represented as the base address of the MPAM feature page in the address space. These are represented here as:
  - MPAMF_BASE_s in the Secure address space
  - MPAMF_BASE_ns in the Non-secure address space.
- If MPAMF_IDR.HAS_RIS is 1, MPAMCFG_PART_SEL.RIS. This field selects a resource to access.
- MPAMCFG_PART_SEL.PARTID. This field selects the PARTID from:
  - The PARTID space
  - The resource instance to be configured.
- The control settings register. When accessed, this register selects which control is being configured for:
  - The PARTID.
  - The PARTID space.
  - The resource instance.

For example, to access the memory bandwidth maximum configuration settings for Secure PARTID 15 on resource instance 2 of an MSC that implements RIS:

1. Secure PARTID 15 must be stored in MPAMCFG_PART_SEL.PARTID at the address MPAMF_BASE_s + `0x0100` and, due to RIS being implemented, the RIS field of that address must be set to 2 to ensure access to the correct resource instance.
2. Once the store has completed, the new maximum fraction of memory bandwidth for Secure PARTID 15 of resource instance 2 must be stored into the MPAMCFG_MBW_MAX_s register of this MSC, found at MPAMF_BASE_s + `0x0208`.

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-391.

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-391.

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 mis-allocation 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 5-57.

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 9-178.

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-392.

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 9-189.

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

- In the fixed-point fractional format, 1.0 is represented as $2^{16} - 1$, 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-391, 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 9-181

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 9-180, 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. For more information, see *About the fixed-point fractional format* on page 9-189.

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. For more information, see *About the fixed-point fractional format* on page 9-189.

Using minimum-bandwidth limit with maximum-bandwidth limit controls

If both minimum-bandwidth limit and maximum-bandwidth limit are implemented, Table 9-1 on page 9-181 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[a].
Above the minimum	Below the maximum limit.	Medium	High requests are serviced first, then compete with other Medium requests[a].
	Above the maximum limit, with HARDLIM clear.	Low	Requests are not serviced if any High or Medium requests are available[a].
	Above the maximum limit, with HARDLIM set.	None	Requests are not serviced.

a. 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 9-189.

MPAMCFG_MBW_MAX, the bandwidth control setting register for maximum-bandwidth limit 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 can 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* on page 9-182). Request or local priorities (*Priority partitioning* on page 9-183) 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-393. 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-393 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 9-183.

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.

In MPAM v0.1 and from MPAM v1.1:

- If MPAMF_IMPL_IDR describes no IMPLEMENTATION DEFINED partitioning controls, MPAMF_IDR.NO_IMPL_PART must be 1.
- If MPAMF_IMPL_IDR describes no IMPLEMENTATION DEFINED monitors, MPAMF_IDR.NO_IMPL_MSMON must be 1.

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 3-38) 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 on page 9-188 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<n>	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 on page 9-189 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%

Table 9-3 Fraction Widths and Hex Representation (continued)

Percentage	16 bits			12 bits			8 bits		
	Hex	Min	Max	Hex	Min	Max	Hex	Min	Max
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 monitors* on page 10-193
- *Cache-storage usage monitors* on page 10-195

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 2^{16} of each type. The PARTID for filtering resource monitors is always a request PARTID, even when PARTID narrowing is implemented.

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 10-198.

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 that configures behavior of the monitor instance.
- A filter register MSMON_CFG_MBWU_FLT that specifies the transfers to be counted. This register has fields for reads, writes, PARTID, PMG, and other criteria.
- A monitor register MSMON_MBWU that 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.
- In MPAM v0.1 and from MPAM v1.1, an optional long monitor register, MSMON_MBWU_L, that contains a count of 44 bits or 63 bits. A NRDY bit is also present in this register, see *Not-Ready Bit* on page 10-196.
- An optional capture register MSMON_MBWU_CAPTURE that 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.
- In MPAM v0.1 and from MPAM v1.1, if MPAMF_MBWUMON_IDR.{HAS_LONG, HAS_CAPTURE} are 1, the MSMON_MBWU_L_CAPTURE register must be implemented.
- A Not-Ready (NRDY) bit (*Not-Ready Bit* on page 10-196) 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 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.
- 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 11-283). 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.

Scaled MBWU count value

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.

Long MBWU counter and capture

In MPAM v0.1 and from MPAM v1.0, there is optional support for 44-bit or 63-bit MBWU counters.

MSMON_MBWU_L is optional and only present when MPAMF_MBWUMON_IDR.HAS_LONG is 1. This indicates that this monitor type supports long counters.

If MPAMF_MBWUMON_IDR.{HAS_LONG, HAS_CAPTURE} are both 1, the MSMON_MBWU_L_CAPTURE register must also be implemented.

The VALUE field of the long registers is never scaled.

The VALUE field of MSMON_MBWU_L and MSMON_MBWU_L_CAPTURE can be implemented either as a 63-bit VALUE field or a 44-bit VALUE field. The 44-bit VALUE field is indicated when MPAMF_MBWUMON_IDR.LWD is 0 and has bits[62:44] of each register as RES0. When MPAMF_MBWUMON_IDR.LWD is 1, the VALUE field of each register is 63 bits.

An overflow occurs in the long counter when the count in the VALUE field exceeds the maximum representable value. This depends on the length of the VALUE field set by MPAMF_MBWUMON_IDR.LWD.

When any instance of the MSMON_MBWU_L counter overflows, the MSMON_CFG_MBWU_CTL.OFLOW_STATUS_L bit is set. If MSMON_CFG_MBWU_CTL.OFLOW_INTR_L is set, this overflow produces an MPAM Overflow interrupt. See *MPAM overflow interrupt* on page 8-167.

When an implementation has both the long counter and the short 31-bit counter, the short counter might overflow when the long counter has not overflowed and produce an MPAM Overflow interrupt. This can be prevented by setting MSMON_CFG_MBWU_CTL.OFLOW_INTR to 0, which disables the overflow interrupt for overflow of the short counter.

The MSMON_CFG_MBWU_CTL.OFLOW_FRZ field is not duplicated, and affects the behaviors of both short and long counters on overflow.

10.2.2 Cache-storage usage monitors

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_CSU_FLT that sets the PARTID and PMG to be monitored.
- A cache-storage usage register MSMON_CSU that reports the amount of storage currently present within the cache allocated by the PARTID and PMG. It is an implementation choice whether MSMON_CSU is implemented as RO or RW.
- A Not-Ready bit in the cache-storage usage register MSMON_CSU that indicates that the value is not accurate. An implementation may set this NRDY bit if the value in the cache-storage usage register is not currently accurate, possibly because it is still being computed. For more information on the Not-Ready bit, see *Not-Ready Bit* on page 10-196.
- 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_CSU_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 might 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 NRDY bit of the cache-storage usage register is 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 10-198). 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.

If the overflow changes the OFLOW_STATUS flag from 0 to 1 and the OFLOW_INTR bit is set, an MPAM overflow interrupt will be signaled if implemented. See also *MPAM overflow interrupt* on page 8-167.

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 on page 10-198) has fields to select the PARTID and PMG to monitor.
- MSMON_CFG_<type>_CTL (Table 10-2 on page 10-198) 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. The PARTID for filtering resource monitors is always a request PARTID, even when PARTID narrowing is implemented.
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	Description	
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.

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

Bits	Name	Description	
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.
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 11-203). 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 11-208 and *MPAM feature page* on page 11-203.

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 11-250).
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 11-250. 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 11-283).
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 11-283. 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 memory-mapped registers and reserved feature page locations* on page 11-205, except for the MPAMCFG_MBW_PBM and MPAMCFG_CPBM as described in *Permitted truncation of an MPAM feature page* on page 11-205.

Permitted truncation of an MPAM feature page

An MPAM feature page may be shortened in only 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.

See *Examples of partial MPAM implementations* on page 11-206 for examples showing MPAMF_*IDR registers in implementations with few MPAM functions.

11.1.6 IMPLEMENTATION DEFINED memory-mapped registers and reserved 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* on page 11-205 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.

11.1.7 Examples of partial MPAM implementations

Most MSCs only implement a fraction of the full MSC MPAM architecture. This section gives examples of partial implementations, some of which have been achieved by partially removing MPAM. The RTL configuration examples are included to illustrate the MMR issues in partial MPAM implementations.

An MSC that has no partitioning or monitoring, only propagation

An MSC that does not implement any resource partitioning or monitor interfaces only requires a few MMRs:

- The minimum required MMRs, as specified in *Minimum required MPAM memory-mapped registers* on page 11-205, must be implemented with the MPAMF_IDR.{PARTID_MAX, PMG_MAX} fields indicating the maximum PARTID that can be propagated.
- All of the HAS_* and NO_* bits in MPAMF_IDR must be zero.
- MPAMF_AIDR must indicate MPAM v1.0 or MPAM v1.1.
- MPAMF_IIDR must identify the implementation.
- MPAMF_SIDR must indicate PARTID_MAX and PMG_MAX for Secure propagation.

No other registers are required.

An MSC when RTL configuration has removed a partitioning control or resource usage monitor

An MSC could be designed to have an RTL configuration option that removes a partitioning control or a resource usage monitor. If so, the HAS_* bits in each of the relevant MPAMF_*IDR registers must be configured to zero when the feature is removed.

An MSC when RTL configuration has removed all MPAM functionality

An MSC could be designed to have an RTL configuration option that removes all of the MPAM functionality. When all of MPAM is deconfigured:

- The minimum required MPAM registers must be present.
- MPAMF_IDR, MPAMF_AIDR and MPAMF_SIDR must all be zero.
- MPAMF_IIDR is permitted to be either all zero or to identify the IP.

Note

Software might still attempt to discover MPAM on this RTL configuration, so the minimum MPAM registers must be present to allow the lack of MPAM function to be discovered.

An MSC when RTL configuration removes a resource instance

An MSC could be designed to have an RTL configuration option that completely removes one or more resource instances. When a resource instance is removed, only the MPAMF_*IDR registers for the corresponding RIS values are changed. All of the ID registers corresponding to that RIS value have each of their RIS-specific fields set to zero. For more information on RIS-specific fields, see *Effects of MPAMCFG_PART_SEL.RIS on values read from other registers* on page 8-159.

11.2 Summary of memory-mapped registers

Table 11-1 on page 11-208 lists the external MPAM registers in order of register offset.

Table 11-1 Index of external MPAM registers ordered by offset

Register	Offset	Length	Description, see
MPAMF_IDR	`0x0000`	64	*MPAMF_IDR, MPAM Features Identification Register* on page 11-221
MPAMF_SIDR	`0x0008`	32	*MPAMF_SIDR, MPAM Features Secure Identification Register* on page 11-249
MPAMF_IIDR	`0x0018`	32	*MPAMF_IIDR, MPAM Implementation Identification Register* on page 11-228
MPAMF_AIDR	`0x0020`	32	*MPAMF_AIDR, MPAM Architecture Identification Register* on page 11-212
MPAMF_IMPL_IDR	`0x0028`	32	*MPAMF_IMPL_IDR, MPAM Implementation-Specific Partitioning Feature Identification Register* on page 11-230
MPAMF_CPOR_IDR	`0x0030`	32	*MPAMF_CPOR_IDR, MPAM Features Cache Portion Partitioning ID register* on page 11-216
MPAMF_CCAP_IDR	`0x0038`	32	*MPAMF_CCAP_IDR, MPAM Features Cache Capacity Partitioning ID register* on page 11-214
MPAMF_MBW_IDR	`0x0040`	32	*MPAMF_MBW_IDR, MPAM Memory Bandwidth Partitioning Identification Register* on page 11-232
MPAMF_PRI_IDR	`0x0048`	32	*MPAMF_PRI_IDR, MPAM Priority Partitioning Identification Register* on page 11-246
MPAMF_PARTID_NRW_IDR	`0x0050`	32	*MPAMF_PARTID_NRW_IDR, MPAM PARTID Narrowing ID register* on page 11-244
MPAMF_MSMON_IDR	`0x0080`	32	*MPAMF_MSMON_IDR, MPAM Resource Monitoring Identification Register* on page 11-240
MPAMF_CSUMON_IDR	`0x0088`	32	*MPAMF_CSUMON_IDR, MPAM Features Cache Storage Usage Monitoring ID register* on page 11-218
MPAMF_MBWUMON_IDR	`0x0090`	32	*MPAMF_MBWUMON_IDR, MPAM Features Memory Bandwidth Usage Monitoring ID register* on page 11-236
MPAMF_ERR_MSI_MPAM	`0x00DC`	32	*MPAMF_ERR_MSI_MPAM, MPAM Error MSI Write MPAM Information Register* on page 11-357
MPAMF_ERR_MSI_ADDR_L	`0x00E0`	32	*MPAMF_ERR_MSI_ADDR_L, MPAM Error MSI Low-part Address Register* on page 11-350
MPAMF_ERR_MSI_ADDR_H	`0x00E4`	32	*MPAMF_ERR_MSI_ADDR_H, MPAM Error MSI High-part Address Register* on page 11-348
MPAMF_ERR_MSI_DATA	`0x00E8`	32	*MPAMF_ERR_MSI_DATA, MPAM Error MSI Data Register* on page 11-355
MPAMF_ERR_MSI_ATTR	`0x00EC`	32	*MPAMF_ERR_MSI_ATTR, MPAM Error MSI Write Attributes Register* on page 11-352
MPAMF_ECR	`0x00F0`	32	*MPAMF_ECR, MPAM Error Control Register* on page 11-346
MPAMF_ESR	`0x00F8`	64	*MPAMF_ESR, MPAM Error Status Register* on page 11-359

Table 11-1 Index of external MPAM registers ordered by offset (continued)

Register	Offset	Length	Description, see
MPAMCFG_PART_SEL	`0x0100`	32	*MPAMCFG_PART_SEL, MPAM Partition Configuration Selection Register* on page 11-277
MPAMCFG_CMAX	`0x0108`	32	*MPAMCFG_CMAX, MPAM Cache Maximum Capacity Partition Configuration Register* on page 11-251
MPAMCFG_MBW_MIN	`0x0200`	32	*MPAMCFG_MBW_MIN, MPAM Memory Bandwidth Minimum Partition Configuration Register* on page 11-264
MPAMCFG_MBW_MAX	`0x0208`	32	*MPAMCFG_MBW_MAX, MPAM Memory Bandwidth Maximum Partition Configuration Register* on page 11-261
MPAMCFG_MBW_WINWD	`0x0220`	32	*MPAMCFG_MBW_WINWD, MPAM Memory Bandwidth Partitioning Window Width Configuration Register* on page 11-274
MPAMCFG_PRI	`0x0400`	32	*MPAMCFG_PRI, MPAM Priority Partition Configuration Register* on page 11-280
MPAMCFG_MBW_PROP	`0x0500`	32	*MPAMCFG_MBW_PROP, MPAM Memory Bandwidth Proportional Stride Partition Configuration Register* on page 11-271
MPAMCFG_INTPARTID	`0x0600`	32	*MPAMCFG_INTPARTID, MPAM Internal PARTID Narrowing Configuration Register* on page 11-258
MSMON_CFG_MON_SEL	`0x0800`	32	*MSMON_CFG_MON_SEL, MPAM Monitor Instance Selection Register* on page 11-303
MSMON_CAPT_EVNT	`0x0808`	32	*MSMON_CAPT_EVNT, MPAM Capture Event Generation Register* on page 11-284
MSMON_CFG_CSU_FLT	`0x0810`	32	*MSMON_CFG_CSU_FLT, MPAM Memory System Monitor Configure Cache Storage Usage Monitor Filter Register* on page 11-291
MSMON_CFG_CSU_CTL	`0x0818`	32	*MSMON_CFG_CSU_CTL, MPAM Memory System Monitor Configure Cache Storage Usage Monitor Control Register* on page 11-287
MSMON_CFG_MBWU_FLT	`0x0820`	32	*MSMON_CFG_MBWU_FLT, MPAM Memory System Monitor Configure Memory Bandwidth Usage Monitor Filter Register* on page 11-299
MSMON_CFG_MBWU_CTL	`0x0828`	32	*MSMON_CFG_MBWU_CTL, MPAM Memory System Monitor Configure Memory Bandwidth Usage Monitor Control Register* on page 11-294
MSMON_CSU	`0x0840`	32	*MSMON_CSU, MPAM Cache Storage Usage Monitor Register* on page 11-306
MSMON_CSU_CAPTURE	`0x0848`	32	*MSMON_CSU_CAPTURE, MPAM Cache Storage Usage Monitor Capture Register* on page 11-309
MSMON_CSU_OFSR	`0x0858`	32	*MSMON_CSU_OFSR, MPAM CSU Monitor Overflow Status Register* on page 11-312
MSMON_MBWU	`0x0860`	32	*MSMON_MBWU, MPAM Memory Bandwidth Usage Monitor Register* on page 11-315
MSMON_MBWU_CAPTURE	`0x0868`	32	*MSMON_MBWU_CAPTURE, MPAM Memory Bandwidth Usage Monitor Capture Register* on page 11-318

Table 11-1 Index of external MPAM registers ordered by offset (continued)

Register	Offset	Length	Description, see
MSMON_MBWU_L	`0x0880`	64	*MSMON_MBWU_L, MPAM Long Memory Bandwidth Usage Monitor Register* on page 11-321
MSMON_MBWU_L_CAPTURE	`0x0890`	64	*MSMON_MBWU_L_CAPTURE, MPAM Long Memory Bandwidth Usage Monitor Capture Register* on page 11-324
MSMON_MBWU_OFSR	`0x0898`	32	*MSMON_MBWU_OFSR, MPAM MBWU Monitor Overflow Status Register* on page 11-327
MSMON_OFLOW_MSI_MPAM	`0x08DC`	32	*MSMON_OFLOW_MSI_MPAM, MPAM Monitor Overflow MSI Write MPAM Information Register* on page 11-339
MSMON_OFLOW_MSI_ADDR_L	`0x08E0`	32	*MSMON_OFLOW_MSI_ADDR_L, MPAM Monitor Overflow MSI Low-part Address Register* on page 11-332
MSMON_OFLOW_MSI_ADDR_H	`0x08E4`	32	*MSMON_OFLOW_MSI_ADDR_H, MPAM Monitor Overflow MSI Write High-part Address Register* on page 11-330
MSMON_OFLOW_MSI_DATA	`0x08E8`	32	*MSMON_OFLOW_MSI_DATA, MPAM Monitor Overflow MSI Write Data Register* on page 11-337
MSMON_OFLOW_MSI_ATTR	`0x08EC`	32	*MSMON_OFLOW_MSI_ATTR, MPAM Monitor Overflow MSI Write Attributes Register* on page 11-334
MSMON_OFLOW_SR	`0x08F0`	32	*MSMON_OFLOW_SR, MPAM Monitor Overflow Status Register* on page 11-342
MPAMCFG_CPBM<n>	`0x1000`	32	*MPAMCFG_CPBM<n>, MPAM Cache Portion Bitmap Partition Configuration Register, n = 0 - 1023* on page 11-254
MPAMCFG_MBW_PBM<n>	`0x2000`	32	*MPAMCFG_MBW_PBM<n>, MPAM Bandwidth Portion Bitmap Partition Configuration Register, n = 0 - 127* on page 11-267

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

Identifies the version of the MPAM architecture that this MSC implements.

Note: The following values are defined for bits [7:0]:

- `0x01` == MPAM architecture v0.1
- `0x10` == MPAM architecture v1.0
- `0x11` == MPAM architecture v1.1

Configurations

The power domain of MPAMF_AIDR is IMPLEMENTATION DEFINED.

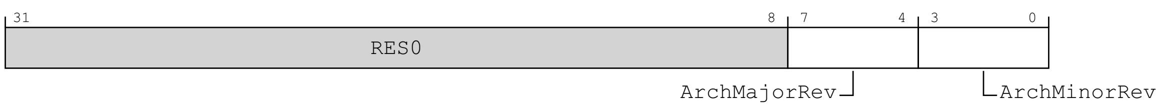
This register is present only when FEAT_MPAM is implemented. Otherwise, direct accesses to MPAMF_AIDR are RES0.

The power and reset domain of each MSC component is specific to that component.

Attributes

MPAMF_AIDR is a 32-bit register.

Field descriptions



Bits [31:8]

Reserved, RES0.

ArchMajorRev, bits [7:4]

Major revision of the MPAM architecture implemented by the MSC.

This table shows the only valid combinations of MPAM version numbers in an MSC. FORCE_NS functionality is only available in MPAM v0.1.

ArchMajorRev	ArchMinorRev	MPAMv	Available
0	0		None.
0	1	v0.1	MPAMv1.0 + MPAMv1.1 + FORCE_NS
1	0	v1.0	MPAMv1.0
1	1	v1.1	MPAMv1.0 + MPAMv1.1 - FORCE_NS

Use of MPAMv0.1 in MSCs is restricted to limited circumstances. The MSC must be able to initiate requests in the Secure address space which have MPAM PARTID forced to the Non-secure space with that forcing not controllable or observable by the software that configures the device for Secure requests. Please contact Arm before setting MPAMF_AIDR to report MPAMv0.1.

ArchMinorRev, bits [3:0]

Minor revision of the MPAM architecture implemented by the MSC.

See the table in the description of the ArchMajorRev field in this register.

Accessing the MPAMF_AIDR:

This register is within the MPAM feature page memory frames.

In a system that supports Secure, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps.

MPAMF_AIDR is read-only.

MPAMF_AIDR must be readable from the Secure, Non-secure, Root, and Realm MPAM feature pages.

MPAMF_AIDR must have the same contents in the Secure, Non-secure, Root, and Realm MPAM feature pages.

MPAMF_AIDR can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_s	`0x0020`	MPAMF_AIDR_s

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_ns

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_rt	`0x0020`	MPAMF_AIDR_rt

This interface is accessible as follows:

- When FEAT_RME is implemented accesses to this register are RO.

MPAMF_AIDR can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rl	`0x0020`	MPAMF_AIDR_rl

This interface is accessible as follows:

- When FEAT_RME is implemented 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

Indicates the number of fractional bits in MPAMCFG_CMAX.CMAX.

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. MPAMF_CCAP_IDR_rt indicates the number of fractional bits in the Root cache capacity control settings register field, MPAMCFG_CMAX.CMAX. MPAMF_CCAP_IDR_rl indicates the number of fractional bits in the Realm cache capacity control settings register field, MPAMCFG_CMAX.CMAX.

When MPAMF_IDR.HAS_RIS is 1, some fields in this register give information for the resource instance selected by MPAMCFG_PART_SEL.RIS. The description of every field that is affected by MPAMCFG_PART_SEL.RIS has information within the field description.

Configurations

The power domain of MPAMF_CCAP_IDR is IMPLEMENTATION DEFINED.

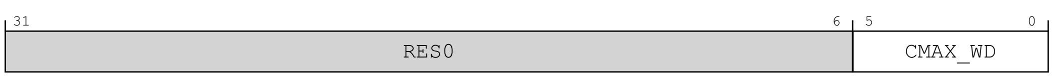
This register is present only when FEAT_MPAM is implemented and MPAMF_IDR.HAS_CCAP_PART == 1. Otherwise, direct accesses to MPAMF_CCAP_IDR are RES0.

The power and reset domain of each MSC component is specific to that component.

Attributes

MPAMF_CCAP_IDR is a 32-bit register.

Field descriptions



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.

If RIS is implemented, this field indicates the number of fractional bits in the cache capacity partitioning control for the resource instance selected by MPAMCFG_PART_SEL.RIS.

Accessing the MPAMF_CCAP_IDR:

This register is within the MPAM feature page memory frames. In a system that supports Secure, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps.

MPAMF_CCAP_IDR is read-only.

MPAMF_CCAP_IDR must be readable from the Non-secure, Secure, Root, and Realm MPAM feature pages.

MPAMF_CCAP_IDR is permitted to have the same contents when read from the Secure, Non-secure, Root, and Realm MPAM feature pages unless the register contents are different for the different versions:

- MPAMF_CCAP_IDR_s is permitted to have either the same or different contents to MPAMF_CCAP_IDR_ns, MPAMF_CCAP_IDR_rt, or MPAMF_CCAP_IDR_rl.

- MPAMF_CCAP_IDR_ns is permitted to have either the same or different contents to MPAMF_CCAP_IDR_rt or MPAMF_CCAP_IDR_rl.
- MPAMF_CCAP_IDR_rt is permitted to have either the same or different contents to MPAMF_CCAP_IDR_rl.

There must be separate registers in the Secure (MPAMF_CCAP_IDR_s), Non-secure (MPAMF_CCAP_IDR_ns), Root (MPAMF_CCAP_IDR_rt), and Realm (MPAMF_CCAP_IDR_rl) MPAM feature pages.

When MPAMF_IDR.HAS_RIS is 1, MPAMF_CCAP_IDR shows the configuration of cache capacity partitioning for the cache resource instance selected by MPAMCFG_PART_SEL.RIS. Fields that mention RIS in their field descriptions have values that track the implemented properties of the resource instance. Fields that do not mention RIS are constant across all resource instances.

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.

MPAMF_CCAP_IDR can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rt	0x0038	MPAMF_CCAP_IDR_rt

This interface is accessible as follows:

- When FEAT_RME is implemented accesses to this register are RO.

MPAMF_CCAP_IDR can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rl	0x0038	MPAMF_CCAP_IDR_rl

This interface is accessible as follows:

- When FEAT_RME is implemented 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

Indicates the number of bits in MPAMCFG_CPBM<n>.

MPAMF_CPOR_IDR_s indicates the number of bits in the Secure instance of MPAMCFG_CPBM<n>. MPAMF_CPOR_IDR_ns indicates the number of bits in the Non-secure instance of MPAMCFG_CPBM<n>. MPAMF_CPOR_IDR_rt indicates the number of bits in the Root instance of MPAMCFG_CPBM<n>. MPAMF_CPOR_IDR_rl indicates the number of bits in the Realm instance of MPAMCFG_CPBM<n>.

When MPAMF_IDR.HAS_RIS is 1, some fields in this register give information for the resource instance selector, MPAMCFG_PART_SEL.RIS. The description of every field that is affected by MPAMCFG_PART_SEL.RIS has information within the field description.

Configurations

The power domain of MPAMF_CPOR_IDR is IMPLEMENTATION DEFINED.

This register is present only when FEAT_MPAM is implemented and MPAMF_IDR.HAS_CPOR_PART == 1. Otherwise, direct accesses to MPAMF_CPOR_IDR are RES0.

The power and reset domain of each MSC component is specific to that component.

Attributes

MPAMF_CPOR_IDR is a 32-bit register.

Field descriptions

31 … 16	15 … 0
RES0	CPBM_WD

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<n>.

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.

If RIS is implemented, this field indicates the number bits in the cache portion bitmap for the resource instance selected by MPAMCFG_PART_SEL.RIS.

Accessing the MPAMF_CPOR_IDR:

This register is within the MPAM feature page memory frames. In a system that supports Secure, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps.

MPAMF_CPOR_IDR is read-only.

MPAMF_CPOR_IDR must be readable from the Non-secure, Secure, Root, and Realm MPAM feature pages.

MPAMF_CPOR_IDR is permitted to have the same contents when read from the Secure, Non-secure, Root, and Realm MPAM feature pages unless the register contents are different for the different versions:

- MPAMF_CPOR_IDR_s is permitted to have either the same or different contents to MPAMF_CPOR_IDR_ns, MPAMF_CPOR_IDR_rt, or MPAMF_CPOR_IDR_rl.

- MPAMF_CPOR_IDR_ns is permitted to have either the same or different contents to MPAMF_CPOR_IDR_rt or MPAMF_CPOR_IDR_rl.
- MPAMF_CPOR_IDR_rt is permitted to have either the same or different contents to MPAMF_CPOR_IDR_rl.

There must be separate registers in the Secure (MPAMF_CPOR_IDR_s), Non-secure (MPAMF_CPOR_IDR_ns), Root (MPAMF_CPOR_IDR_rt), and Realm (MPAMF_CPOR_IDR_rl) MPAM feature pages.

When MPAMF_IDR.HAS_RIS is 1, MPAMF_CPOR_IDR shows the configuration of cache portion partitioning for the cache resource instance selected by MPAMCFG_PART_SEL.RIS. Fields that mention RIS in their field descriptions have values that track the implemented properties of the resource instance. Fields that do not mention RIS are constant across all resource instances.

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.

MPAMF_CPOR_IDR can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rt	0x0030	MPAMF_CPOR_IDR_rt

This interface is accessible as follows:

- When FEAT_RME is implemented accesses to this register are RO.

MPAMF_CPOR_IDR can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rl	0x0030	MPAMF_CPOR_IDR_rl

This interface is accessible as follows:

- When FEAT_RME is implemented 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

Indicates the number of cache storage usage monitor instances and other properties of the CSU monitoring.

MPAMF_CSUMON_IDR_s indicates the number and properties of Secure cache storage usage monitoring. MPAMF_CSUMON_IDR_ns indicates the number and properties of Non-secure cache storage usage monitoring. MPAMF_CSUMON_IDR_rt indicates the number and properties of Root cache storage usage monitoring. MPAMF_CSUMON_IDR_rl indicates the number and properties of Realm cache storage usage monitoring.

If MPAMF_IDR.HAS_RIS is 1, fields that mention RIS must reflect the properties of the resource instance currently selected by MPAMCFG_PART_SEL.RIS. Fields that do not mention RIS are constant across all resource instances.

Configurations

The power domain of MPAMF_CSUMON_IDR is IMPLEMENTATION DEFINED.

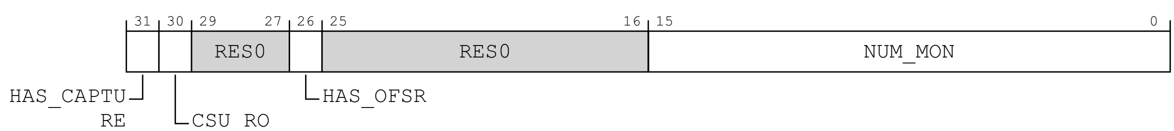
This register is present only when FEAT_MPAM is implemented, MPAMF_IDR.HAS_MSMON == 1 and MPAMF_MSMON_IDR.MSMON_CSU == 1. Otherwise, direct accesses to MPAMF_CSUMON_IDR are RES0.

The power and reset domain of each MSC component is specific to that component.

Attributes

MPAMF_CSUMON_IDR is a 32-bit register.

Field descriptions



HAS_CAPTURE, bit [31]

The 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 the CSU monitor.

`0b1` The MSMON_CSU_CAPTURE register is implemented and the CSU monitor supports the capture event behavior.

If RIS is implemented, this field indicates that CSU monitor capture is implemented for the resource instance selected by MPAMCFG_PART_SEL.RIS.

CSU_RO, bit [30]

The implementation of MSMON_CSU is read-only.

`0b0` MSMON_CSU is read/write.

`0b1` MSMON_CSU is read-only.

If RIS is implemented, this field indicates that the MSMON_CSU monitor register is read-only for the resource instance selected by MPAMCFG_PART_SEL.RIS.

Bits [29:27]

Reserved, RES0.

HAS_OFSR, bit [26]

When FEAT_MPAMv0p1 is implemented or FEAT_MPAMv1p1 is implemented:

The CSU monitor overflow status bitmap register, MSMON_CSU_OFSR, is implemented.

`0b0` MSMON_CSU_OFSR register is not implemented.

`0b1` MSMON_CSU_OFSR register is implemented.

If RIS is implemented, this field indicates that CSU monitor overflow status bitmap register is implemented for the resource instance selected by MPAMCFG_PART_SEL.RIS.

Otherwise:

Reserved, RES0.

Bits [25:16]

Reserved, RES0.

NUM_MON, bits [15:0]

The number of cache storage usage monitor instances implemented.

The largest MSMON_CFG_MON_SEL.MON_SEL value is NUM_MON minus 1.

If RIS is implemented, this field indicates the number of CSU monitor instances implemented for the resource instance selected by MPAMCFG_PART_SEL.RIS.

Accessing the MPAMF_CSUMON_IDR:

This register is within the MPAM feature page memory frames. In a system that supports Secure, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps.

MPAMF_CSUMON_IDR is read-only.

MPAMF_CSUMON_IDR must be readable from the Non-secure, Secure, Root, and Realm MPAM feature pages.

MPAMF_CSUMON_IDR is permitted to have the same contents when read from the Secure, Non-secure, Root, and Realm MPAM feature pages unless the register contents are different for the different versions:

- MPAMF_CSUMON_IDR_s is permitted to have either the same or different contents to MPAMF_CSUMON_IDR_ns, MPAMF_CSUMON_IDR_rt, or MPAMF_CSUMON_IDR_rl.
- MPAMF_CSUMON_IDR_ns is permitted to have either the same or different contents to MPAMF_CSUMON_IDR_rt or MPAMF_CSUMON_IDR_rl.
- MPAMF_CSUMON_IDR_rt is permitted to have either the same or different contents to MPAMF_CSUMON_IDR_rl.

There must be separate registers in the Secure (MPAMF_CSUMON_IDR_s), Non-secure (MPAMF_CSUMON_IDR_ns), Root (MPAMF_CSUMON_IDR_rt), and Realm (MPAMF_CSUMON_IDR_rl) MPAM feature pages.

When MPAMF_IDR.HAS_RIS is 1, MPAMF_CSUMON_IDR shows the configuration of cache storage usage monitoring for the cache resource instance selected by MPAMCFG_PART_SEL.RIS. Fields that mention RIS in their field descriptions have values that track the implemented properties of the resource instance. Fields that do not mention RIS are constant across all resource instances.

Access to MPAMF_CSUMON_IDR is not affected by MSMON_CFG_MON_SEL.RIS.

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.

MPAMF_CSUMON_IDR can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rt	0x0088	MPAMF_CSUMON_IDR_rt

This interface is accessible as follows:

- When FEAT_RME is implemented accesses to this register are RO.

MPAMF_CSUMON_IDR can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rl	0x0088	MPAMF_CSUMON_IDR_rl

This interface is accessible as follows:

- When FEAT_RME is implemented accesses to this register are RO.

11.3.5 MPAMF_IDR, MPAM Features Identification Register

The MPAMF_IDR characteristics are:

Purpose

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. MPAMF_IDR_rt indicates the MPAM features accessed from the Root MPAM feature page. MPAMF_IDR_rl indicates the MPAM features accessed from the Realm MPAM feature page.

When MPAMF_IDR.HAS_RIS is 1, some fields in this register give information for the resource instance selected by MPAMCFG_PART_SEL.RIS. The description of every field that is affected by MPAMCFG_PART_SEL.RIS has that information within the field description.

Configurations

The power domain of MPAMF_IDR is IMPLEMENTATION DEFINED.

This register is present only when FEAT_MPAM is implemented. Otherwise, direct accesses to MPAMF_IDR are RES0.

MPAMF_IDR is 64-bit register when MPAM v0.1 or v1.1 is implemented.

Otherwise, MPAMF_IDR is a 32-bit register.

The power and reset domain of each MSC component is specific to that component.

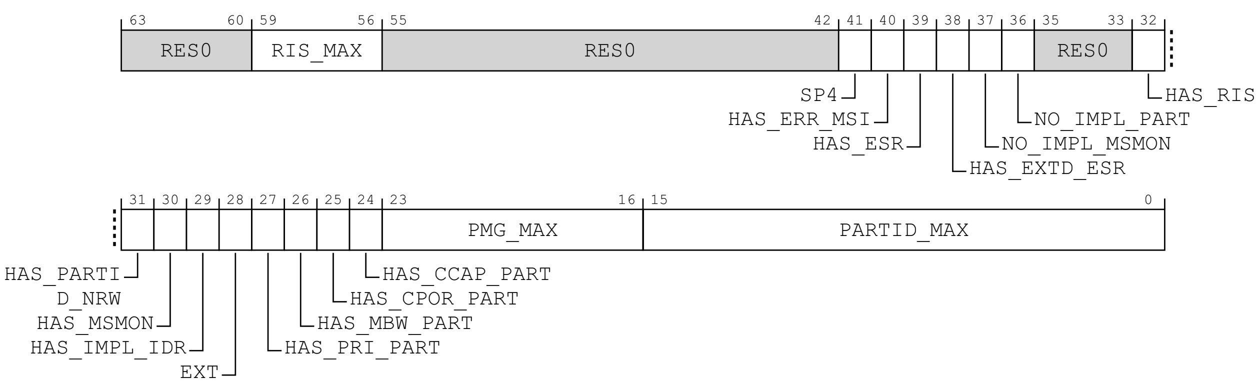
Attributes

MPAMF_IDR is a:

- 64-bit register when FEAT_MPAMv0p1 is implemented or FEAT_MPAMv1p1 is implemented
- 32-bit register otherwise

Field descriptions

When FEAT_MPAMv0p1 is implemented or FEAT_MPAMv1p1 is implemented:



Bits [63:60]

Reserved, RES0.

RIS_MAX, bits [59:56]

When MPAMF_IDR.EXT == 1 and MPAMF_IDR.HAS_RIS == 1:

Maximum RIS value supported in MPAMCFG_PART_SEL. Must be `0b0000` if MPAMF_IDR.HAS_RIS == 0.

Otherwise:

Reserved, RES0.

Bits [55:44]

Reserved, RES0.

SP4, bit [41]

When FEAT_RME is implemented:

Indicates whether this MSC supports 4 PARTID spaces.

`0b0` This MSC supports two PARTID spaces.

`0b1` This MSC supports four PARTID spaces.

This field must read the same in each instance of this register and for any value in MPAMCFG_PART_SEL.RIS.

Otherwise:

Reserved, RES0.

HAS_ERR_MSI, bit [40]

When MPAMF_IDR.EXT == 1:

Has support for MSI writes to signal MPAM error interrupts. These registers are implemented: MPAMF_ERR_MSI_ADDR_L, MPAMF_ERR_MSI_ADDR_H, MPAMF_ERR_MSI_ATTR, MPAMF_ERR_MSI_DATA, and MPAMF_ERR_MSI_MPAM.

`0b0` MPAMF_ERR_MSI_ADDR_L, MPAMF_ERR_MSI_ADDR_H, MPAMF_ERR_MSI_ATTR, MPAMF_ERR_MSI_DATA, and MPAMF_ERR_MSI_MPAM registers are not implemented.

`0b1` MPAMF_ERR_MSI_ADDR_L, MPAMF_ERR_MSI_ADDR_H, MPAMF_ERR_MSI_ATTR, MPAMF_ERR_MSI_DATA, and MPAMF_ERR_MSI_MPAM are implemented and can be used to generate writes to signal error interrupts.

If MPAMF_IDR.HAS_ESR is 0, this bit must also be 0.

Otherwise:

Reserved, RES0.

HAS_ESR, bit [39]

When MPAMF_IDR.EXT == 1:

MPAMF_ESR is implemented.

`0b0` MPAMF_ESR, MPAMF_ECR, and MPAM error handling are not implemented.

`0b1` MPAMF_ESR, MPAMF_ECR, and MPAM error handling are implemented.

If an MSC cannot encounter any of the error conditions listed in Errors in MSCs, both the MPAMF_ESR and MPAMF_ECR must be RAZ/WI.

Otherwise:

Reserved, RES0.

HAS_EXTD_ESR, bit [38]

When MPAMF_IDR.EXT == 1:

MPAMF_ESR is 64 bits.

`0b0` MPAMF_ESR is 32 bits.

`0b1` MPAMF_ESR is 64 bits.

When MPAMF_IDR.HAS_RIS and MPAMF_IDR.HAS_ESR, this field must be 1.

Otherwise:

Reserved, RES0.

NO_IMPL_MSMON, bit [37]

When MPAMF_IDR.EXT == 1 and MPAMF_IDR.HAS_IMPL_IDR == 1:

MPAMF_IMPL_IDR defines no IMPLEMENTATION DEFINED resource monitors.

`0b0` MPAMF_IMPL_IDR defines at least one IMPLEMENTATION DEFINED resource monitor.

`0b1` MPAMF_IMPL_IDR does not define any IMPLEMENTATION DEFINED resource monitors.

If RIS is implemented, this field indicates the presence of IMPLEMENTATION DEFINED resource monitors described in MPAMF_IMPL_IDR for the selected resource instance.

Otherwise:

Reserved, RES0.

NO_IMPL_PART, bit [36]

When MPAMF_IDR.EXT == 1 and MPAMF_IDR.HAS_IMPL_IDR == 1:

MPAMF_IMPL_IDR defines no IMPLEMENTATION DEFINED resource controls.

`0b0` MPAMF_IMPL_IDR defines at least one IMPLEMENTATION DEFINED resource control.

`0b1` MPAMF_IMPL_IDR does not define any IMPLEMENTATION DEFINED resource controls.

If RIS is implemented, this field indicates the presence of IMPLEMENTATION DEFINED resource controls described in MPAMF_IMPL_IDR for the selected resource instance.

Otherwise:

Reserved, RES0.

Bits [35:33]

Reserved, RES0.

HAS_RIS, bit [32]

When MPAMF_IDR.EXT == 1:

Has resource instance selector. Indicates that MPAMCFG_PART_SEL contains the RIS field that selects a resource instance to control.

`0b0` MPAMCFG_PART_SEL does not implement the MPAMCFG_PART_SEL.RIS field or multiple resource instance support.

`0b1` MPAMCFG_PART_SEL implements the MPAMCFG_PART_SEL.RIS field and MPAM resource instance numbers up to and including MPAMF_IDR.RIS_MAX.

Otherwise:

Reserved, RES0.

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.

EXT, bit [28]

When FEAT_MPAMv0p1 is implemented or FEAT_MPAMv1p1 is implemented:

Extended MPAMF_IDR.

`0b0` MPAMF_IDR has no defined bits in [63:32]. The register is effectively 32 bits.

`0b1` MPAMF_IDR has bits defined in [63:32]. The register is 64-bits.

Otherwise:

Reserved, RES0.

HAS_PRI_PART, bit [27]

Has priority partitioning. Indicates that MPAM priority partitioning is implemented and MPAMF_PRI_IDR exists.

`0b0` Does not support priority partitioning or have MPAMF_PRI_IDR.

`0b1` Has priority partitioning and MPAMF_PRI_IDR.

If RIS is implemented, this field indicates the presence of priority partitioning resource controls as described in MPAMF_PRI_IDR for the selected resource instance.

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.

If RIS is implemented, this field indicates the presence of memory bandwidth partitioning resource controls as described in MPAMF_MBW_IDR for the selected resource instance.

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<n> registers.

`0b1` Has MPAMF_CPOR_IDR and MPAMCFG_CPBM<n> registers.

If RIS is implemented, this field indicates the presence of cache portion partitioning resource controls as described in MPAMF_CPOR_IDR for the selected resource instance.

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.

If RIS is implemented, this field indicates the presence of cache capacity partitioning resource controls as described in MPAMF_CPOR_IDR for the selected resource instance.

PMG_MAX, bits [23:16]

Maximum supported value of PMG.

The value of this field is permitted to vary between the instances of MPAM_IDR, each reporting the maximum supported PMG value in the PARTID space associated with that instance.

In MPAMF_IDR_s, this field is permitted to report the maximum PMG value for the Non-secure PARTID space or for the Secure PARTID space. The maximum PMG value for the Secure PARTID space can be read from MPAMF_SIDR.PMG_MAX.

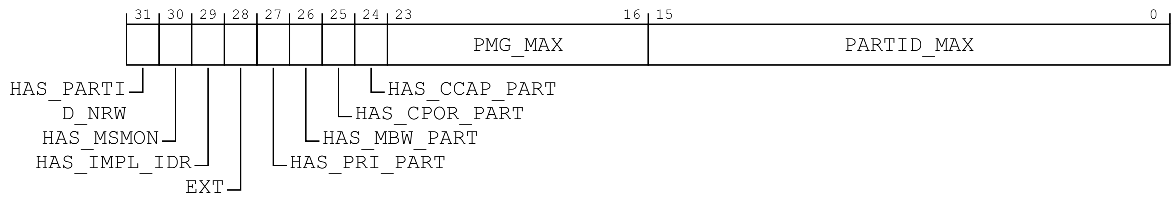
PARTID_MAX, bits [15:0]

Maximum supported value of PARTID.

The value of this field is permitted to vary between the instances of MPAM_IDR, each reporting the maximum supported PARTID value in the PARTID space associated with that instance.

In MPAMF_IDR_s this field is permitted to report the maximum PARTID value for the Non-secure PARTID space or for the Secure PARTID space. The maximum PARTID value for the Secure PARTID space can be read from MPAMF_SIDR.PARTID_MAX.

Otherwise:



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.

EXT, bit [28]

When FEAT_MPAMv0p1 is implemented or FEAT_MPAMv1p1 is implemented:

Extended MPAMF_IDR.

`0b0` MPAMF_IDR has no defined bits in [63:32]. The register is effectively 32 bits.

`0b1` MPAMF_IDR has bits defined in [63:32]. The register is 64-bits.

Otherwise:

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<n> registers.

`0b1` Has MPAMF_CPOR_IDR and MPAMCFG_CPBM<n> 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, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps.

MPAMF_IDR is read-only.

MPAMF_IDR must be readable from the Non-secure, Secure, Root, and Realm MPAM feature pages.

MPAMF_IDR is permitted to have the same contents when read from the Secure, Non-secure, Root, and Realm MPAM feature pages unless the register contents are different for the different versions:

- MPAMF_IDR_s is permitted to have either the same or different contents to MPAMF_IDR_ns, MPAMF_IDR_rt, or MPAMF_IDR_rl.
- MPAMF_IDR_ns is permitted to have either the same or different contents to MPAMF_IDR_rt or MPAMF_IDR_rl.
- MPAMF_IDR_rt is permitted to have either the same or different contents to MPAMF_IDR_rl.

There must be separate registers in the Secure (MPAMF_IDR_s), Non-secure (MPAMF_IDR_ns), Root (MPAMF_IDR_rt), and Realm (MPAMF_IDR_rl) MPAM feature pages.

When MPAMF_IDR.HAS_RIS is 1, MPAMF_IDR shows the configuration of MSC MPAM for the resource instance selected by MPAMCFG_PART_SEL.RIS. Fields that mention RIS in their field descriptions have values that track the implemented properties of the resource instance. Fields that do not mention RIS are constant across all resource instances.

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.

MPAMF_IDR can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rt	`0x0000`	MPAMF_IDR_rt

This interface is accessible as follows:

- When FEAT_RME is implemented accesses to this register are RO.

MPAMF_IDR can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rl	`0x0000`	MPAMF_IDR_rl

This interface is accessible as follows:

- When FEAT_RME is implemented accesses to this register are RO.

11.3.6 MPAMF_IIDR, MPAM Implementation Identification Register

The MPAMF_IIDR characteristics are:

Purpose

Uniquely identifies the MSC implementation by the combination of implementer, product ID, variant, and revision.

Configurations

The power domain of MPAMF_IIDR is IMPLEMENTATION DEFINED.

This register is present only when FEAT_MPAM is implemented. Otherwise, direct accesses to MPAMF_IIDR are RES0.

The power and reset domain of each MSC component is specific to that component.

Attributes

MPAMF_IIDR is a 32-bit register.

Field descriptions

31 – 20	19 – 16	15 – 12	11 – 0
ProductID	Variant	Revision	Implementer

ProductID, bits [31:20]

The MSC implementer as identified in the MPAMF_IIDR.Implementer field must assure each product has a unique ProductID from any other with the same Implementer value.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is RO.

Variant, bits [19:16]

This field distinguishes product variants or major revisions of the product.

Note

Implementations of ProductID with differing software interfaces are expected to have different values in the MPAMF_IIDR.Variant field.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is RO.

Revision, bits [15:12]

This field distinguishes minor revisions of the product.

Note

This field is intended to differentiate product revisions that are minor changes and are largely software compatible with previous revisions.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is RO.

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, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps.

MPAMF_IIDR is read-only.

MPAMF_IIDR must be readable from the Secure, Non-secure, Root, and Realm MPAM feature pages.

MPAMF_IIDR must have the same contents in the Secure, Non-secure, Root, and Realm MPAM feature pages.

MPAMF_IIDR can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_s	`0x0018`	MPAMF_IIDR_s

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_ns

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_rt	`0x0018`	MPAMF_IIDR_rt

This interface is accessible as follows:

- When FEAT_RME is implemented accesses to this register are RO.

MPAMF_IIDR can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rl	`0x0018`	MPAMF_IIDR_rl

This interface is accessible as follows:

- When FEAT_RME is implemented 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

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. MPAMF_IMPL_IDR_rt indicates IMPLEMENTATION DEFINED partitioning and monitoring features accessed from the Root MPAM feature page. MPAMF_IMPL_IDR_rl indicates those accessed from the Realm MPAM feature page.

If MPAMF_IDR.HAS_RIS is 1, this register gives the implementation-specific features and parameters of the resource instance selected by MPAMCFG_PART_SEL.RIS for any features that are specific to the resource.

Configurations

The power domain of MPAMF_IMPL_IDR is IMPLEMENTATION DEFINED.

This register is present only when FEAT_MPAM is implemented and MPAMF_IDR.HAS_IMPL_IDR == 1. Otherwise, direct accesses to MPAMF_IMPL_IDR are RES0.

The power and reset domain of each MSC component is specific to that component.

Attributes

MPAMF_IMPL_IDR is a 32-bit register.

Field descriptions

31 … 0
IMPLFEAT

IMPLFEAT, bits [31:0]

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.

If RIS is implemented, this register indicates the implementation-specific features and parameters of the resource instance selected by MPAMCFG_PART_SEL.RIS.

Accessing the MPAMF_IMPL_IDR:

This register is within the MPAM feature page memory frames. In a system that supports Secure, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps.

MPAMF_IMPL_IDR is read-only.

MPAMF_IMPL_IDR must be readable from the Non-secure, Secure, Root, and Realm MPAM feature pages.

MPAMF_IMPL_IDR is permitted to have the same contents when read from the Secure, Non-secure, Root, and Realm MPAM feature pages unless the register contents are different for the different versions:

- MPAMF_IMPL_IDR_s is permitted to have either the same or different contents to MPAMF_IMPL_IDR_ns, MPAMF_IMPL_IDR_rt, or MPAMF_IMPL_IDR_rl.
- MPAMF_IMPL_IDR_ns is permitted to have either the same or different contents to MPAMF_IMPL_IDR_rt or MPAMF_IMPL_IDR_rl.

- MPAMF_IMPL_IDR_rt is permitted to have either the same or different contents to MPAMF_IMPL_IDR_rl.

There must be separate registers in the Secure (MPAMF_IMPL_IDR_s), Non-secure (MPAMF_IMPL_IDR_ns), Root (MPAMF_IMPL_IDR_rt), and Realm (MPAMF_IMPL_IDR_rl) MPAM feature pages.

When MPAMF_IDR.HAS_RIS is 1, MPAMF_IMPL_IDR shows the configuration of implementation-specific features for the resource instance selected by MPAMCFG_PART_SEL.RIS. Fields that mention RIS in their field descriptions have values that track the implemented properties of the resource instance. Fields that do not mention RIS are constant across all resource instances.

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.

MPAMF_IMPL_IDR can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rt	0x0028	MPAMF_IMPL_IDR_rt

This interface is accessible as follows:

- When FEAT_RME is implemented accesses to this register are RO.

MPAMF_IMPL_IDR can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rl	0x0028	MPAMF_IMPL_IDR_rl

This interface is accessible as follows:

- When FEAT_RME is implemented 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

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. MPAMF_MBW_IDR_rt indicates bandwidth partitioning features accessed from the Root MPAM feature page. MPAMF_MBW_IDR_rl indicates bandwidth partitioning features accessed from the Realm MPAM feature page.

When MPAMF_IDR.HAS_RIS is 1, some fields in this register give information for the resource instance selected by MPAMCFG_PART_SEL.RIS. The description of every field that is affected by MPAMCFG_PART_SEL.RIS has that information within the field description.

Configurations

The power domain of MPAMF_MBW_IDR is IMPLEMENTATION DEFINED.

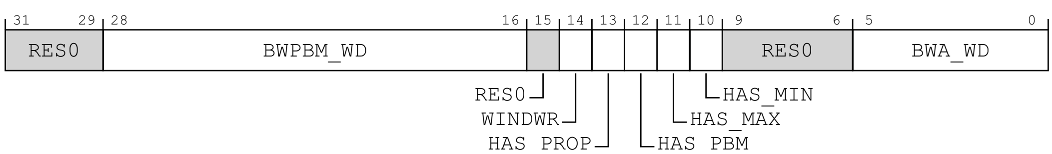
This register is present only when FEAT_MPAM is implemented and MPAMF_IDR.HAS_MBW_PART == 1. Otherwise, direct accesses to MPAMF_MBW_IDR are RES0.

The power and reset domain of each MSC component is specific to that component.

Attributes

MPAMF_MBW_IDR is a 32-bit register.

Field descriptions



Bits [31:29]

Reserved, RES0.

BWPBM_WD, bits [28:16]

Bandwidth portion bitmap width.

The number of bandwidth portion bits in the MPAMCFG_MBW_PBM<n> register array.

If MPAMF_MBW_IDR.HAS_PBM is 1, 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.

If MPAMF_MBW_IDR.HAS_PBM is 0, this field must be ignored by software.

If RIS is implemented, this field indicates the width of the memory bandwidth portion bitmap partitioning control for the resource instance selected by MPAMCFG_PART_SEL.RIS.

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 can be accessed.

`0b0` There is no memory bandwidth proportional stride control and the MPAMCFG_MBW_PROP register is RES0.

`0b1` The proportional stride memory bandwidth partitioning scheme is supported and the MPAMCFG_MBW_PROP register can be accessed.

If RIS is implemented, this field indicates the presence of the memory bandwidth proportional stride partitioning control for the resource instance selected by MPAMCFG_PART_SEL.RIS.

HAS_PBM, bit [12]

Indicates that bandwidth portion partitioning is implemented and the MPAMCFG_MBW_PBM<n> register array can be accessed.

`0b0` There is no memory bandwidth portion control and the MPAMCFG_MBW_PBM<n> is RES0.

`0b1` The memory bandwidth portion allocation scheme exists and the MPAMCFG_MBW_PBM<n> register can be accessed.

If RIS is implemented, this field indicates the presence of the memory bandwidth portion partitioning control for the resource instance selected by MPAMCFG_PART_SEL.RIS.

HAS_MAX, bit [11]

Indicates that this MSC implements maximum bandwidth partitioning and the MPAMCFG_MBW_MAX register can be accessed.

`0b0` There is no maximum memory bandwidth control and the MPAMCFG_MBW_MAX register is RES0.

`0b1` The maximum memory bandwidth allocation scheme is supported and the MPAMCFG_MBW_MAX register can be accessed.

If RIS is implemented, this field indicates the presence of the maximum bandwidth partitioning control for the resource instance selected by MPAMCFG_PART_SEL.RIS.

HAS_MIN, bit [10]

Indicates that this MSC implements minimum bandwidth partitioning and the MPAMCFG_MBW_MIN register can be accessed.

`0b0` There is no minimum memory bandwidth control and the MPAMCFG_MBW_MIN register is RES0.

`0b1` The minimum memory bandwidth allocation scheme is supported and the MPAMCFG_MBW_MIN register can be accessed.

If RIS is implemented, this field indicates the presence of the minimum bandwidth partitioning control for the resource instance selected by MPAMCFG_PART_SEL.RIS.

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.

In any of these bandwidth allocation fields exist, this field must have a value from 1 to 16, inclusive. Otherwise, it is permitted to be 0.

If RIS is implemented, this field indicates the number of implemented bits in the bandwidth allocation control fields for the resource instance selected by MPAMCFG_PART_SEL.RIS.

Accessing the MPAMF_MBW_IDR:

This register is within the MPAM feature page memory frames. In a system that supports Secure, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps.

MPAMF_MBW_IDR is read-only.

MPAMF_MBW_IDR must be readable from the Non-secure, Secure, Root, and Realm MPAM feature pages.

MPAMF_MBW_IDR is permitted to have the same contents when read from the Secure, Non-secure, Root, and Realm MPAM feature pages unless the register contents are different for the different versions:

- MPAMF_MBW_IDR_s is permitted to have either the same or different contents to MPAMF_MBW_IDR_ns, MPAMF_MBW_IDR_rt, or MPAMF_MBW_IDR_rl.
- MPAMF_MBW_IDR_ns is permitted to have either the same or different contents to MPAMF_MBW_IDR_rt or MPAMF_MBW_IDR_rl.
- MPAMF_MBW_IDR_rt is permitted to have either the same or different contents to MPAMF_MBW_IDR_rl.

There must be separate registers in the Secure (MPAMF_MBW_IDR_s), Non-secure (MPAMF_MBW_IDR_ns), Root (MPAMF_MBW_IDR_rt), and Realm (MPAMF_MBW_IDR_rl) MPAM feature pages.

When MPAMF_IDR.HAS_RIS is 1, MPAMF_MBW_IDR shows the configuration of memory bandwidth partitioning for the bandwidth resource instance selected by MPAMCFG_PART_SEL.RIS. Fields that mention RIS in their field descriptions have values that track the implemented properties of the resource instance. Fields that do not mention RIS are constant across all resource instances.

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.

MPAMF_MBW_IDR can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rt	0x0040	MPAMF_MBW_IDR_rt

This interface is accessible as follows:

- When FEAT_RME is implemented accesses to this register are RO.

MPAMF_MBW_IDR can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rl	0x0040	MPAMF_MBW_IDR_rl

This interface is accessible as follows:

- When FEAT_RME is implemented 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

Indicates the number of memory bandwidth usage monitor instances implemented. This register also indicates several properties of MBWU monitoring, including whether the implementation supports capture, scaling, or long counters.

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. MPAMF_MBWUMON_IDR_rt indicates the number of Root memory bandwidth usage monitor instances. MPAMF_MBWUMON_IDR_rl indicates the number of Realm memory bandwidth usage monitor instances.

If MPAMF_IDR.HAS_RIS is 1, fields that mention RIS must reflect the properties of the resource instance currently selected by MPAMCFG_PART_SEL.RIS. Fields that do not mention RIS are constant across all resource instances.

Configurations

The power domain of MPAMF_MBWUMON_IDR is IMPLEMENTATION DEFINED.

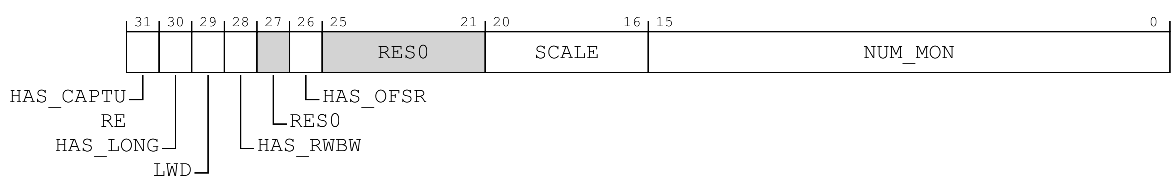
This register is present only when FEAT_MPAM is implemented, MPAMF_IDR.HAS_MSMON == 1 and MPAMF_MSMON_IDR.MSMON_MBWU == 1. Otherwise, direct accesses to MPAMF_MBWUMON_IDR are RES0.

The power and reset domain of each MSC component is specific to that component.

Attributes

MPAMF_MBWUMON_IDR is a 32-bit register.

Field descriptions



HAS_CAPTURE, bit [31]

The 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 the MBWU monitor.

`0b1` The MSMON_MBWU_CAPTURE register is implemented and the MBWU monitor supports the capture event behavior.

If RIS is implemented, this field indicates that MBWU monitor capture is implemented for the resource instance selected by MPAMCFG_PART_SEL.RIS.

If MPAMF_MBWUMON_IDR.HAS_LONG is 1, this also indicates that MSMON_MBWU_L_CAPTURE is implemented.

HAS_LONG, bit [30]

When FEAT_MPAMv0p1 is implemented or FEAT_MPAMv1p1 is implemented:

Indicates whether MSMON_MBWU_L is implemented.

If HAS_CAPTURE is 1, indicates whether MSMON_MBWU_L_CAPTURE is implemented.

`0b0` Does not implement MSMON_MBWU_L or MSMON_MBWU_L_CAPTURE.

`0b1` Implements MSMON_MBWU_L. If HAS_CAPTURE == 1, MSMON_MBWU_L_CAPTURE is also implemented.

If RIS is implemented, this field indicates that the long MBWU monitor is implemented for the resource instance selected by MPAMCFG_PART_SEL.RIS.

If MPAMF_MBWUMON_IDR.HAS_CAPTURE is 1, this also indicates that MSMON_MBWU_L_CAPTURE is implemented.

Otherwise:

Reserved, RES0.

LWD, bit [29]

When FEAT_MPAMv0p1 is implemented or FEAT_MPAMv1p1 is implemented:

Long register VALUE width.

If MPAMF_MBWUMON_IDR.HAS_LONG is 0, MPAMF_MBWUMON_IDR.LWD must also be 0.

`0b0` If MPAMF_MBWUMON_IDR.HAS_LONG is 1, MSMON_MBWU_L has 44-bit VALUE field in bits [43:0]. Bits [62:44] are RES0. If HAS_LONG is 1 and MPAMF_MBWUMON_IDR.HAS_CAPTURE is 1, MSMON_MBWU_L_CAPTURE also has 44-bit VALUE field in bits [43:0].

`0b1` MSMON_MBWU_L has 63-bit VALUE field in bits [62:0]. If MPAMF_MBWUMON_IDR.HAS_CAPTURE == 1, MSMON_MBWU_L_CAPTURE also has 63-bit VALUE field in bits [62:0].

If RIS is implemented, this field indicates the length of the MSMON_MBWU_L.VALUE field implemented for the resource instance selected by MPAMCFG_PART_SEL.RIS.

Otherwise:

Reserved, RES0.

HAS_RWBW, bit [28]

When FEAT_MPAMv0p1 is implemented or FEAT_MPAMv1p1 is implemented:

Read/write bandwidth selection is implemented in MSMON_CFG_MBWU_FLT.

`0b0` Read/write bandwidth selection is not implemented.

`0b1` Read/write bandwidth selection is implemented.

If RIS is implemented, this field indicates whether read/write bandwidth collection selection is available in MSMON_CFG_MBWU_FLT for resource instance selected by MPAMCFG_PART_SEL.RIS.

Otherwise:

Reserved, RES0.

Bit [27]

Reserved, RES0.

HAS_OFSR, bit [26]

When FEAT_MPAMv0p1 is implemented or FEAT_MPAMv1p1 is implemented:

The MBWU monitor overflow status bitmap register, MSMON_MBWU_OFSR, is implemented.

`0b0` MSMON_MBWU_OFSR register is not implemented.

`0b1` MSMON_MBWU_OFSR register is implemented.

If RIS is implemented, this field indicates that MBWU monitor overflow status bitmap register is implemented for the resource instance selected by MPAMCFG_PART_SEL.RIS.

Otherwise:

Reserved, RES0.

Bits [25:21]

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.

If RIS is implemented, this field indicates the scale value for MSMON_MBWU.VALUE field for the resource instance selected by MPAMCFG_PART_SEL.RIS.

NUM_MON, bits [15:0]

The number of memory bandwidth usage monitor instances implemented. The largest monitor instance selector, MSMON_CFG_MON_SEL.MON_SEL, is NUM_MON minus 1.

If RIS is implemented, this field indicates the number of MBWU monitor instances for MSMON_MBWU.VALUE field for the resource instance selected by MPAMCFG_PART_SEL.RIS.

Accessing the MPAMF_MBWUMON_IDR:

This register is within the MPAM feature page memory frames. In a system that supports Secure, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps.

MPAMF_MBWUMON_IDR is read-only.

MPAMF_MBWUMON_IDR must be readable from the Non-secure, Secure, Root, and Realm MPAM feature pages.

MPAMF_MBWUMON_IDR is permitted to have the same contents when read from the Secure, Non-secure, Root, and Realm MPAM feature pages unless the register contents are different for the different versions:

- MPAMF_MBWUMON_IDR_s is permitted to have either the same or different contents to MPAMF_MBWUMON_IDR_ns, MPAMF_MBWUMON_IDR_rt, or MPAMF_MBWUMON_IDR_rl.
- MPAMF_MBWUMON_IDR_ns is permitted to have either the same or different contents to MPAMF_MBWUMON_IDR_rt or MPAMF_MBWUMON_IDR_rl.
- MPAMF_MBWUMON_IDR_rt is permitted to have either the same or different contents to MPAMF_MBWUMON_IDR_rl.

There must be separate registers in the Secure (MPAMF_MBWUMON_IDR_s), Non-secure (MPAMF_MBWUMON_IDR_ns), Root (MPAMF_MBWUMON_IDR_rt), and Realm (MPAMF_MBWUMON_IDR_rl) MPAM feature pages.

When MPAMF_IDR.HAS_RIS is 1, MPAMF_MBWUMON_IDR shows the configuration of memory bandwidth monitoring for the bandwidth resource instance selected by MPAMCFG_PART_SEL.RIS. Fields that mention RIS in their field descriptions have values that track the implemented properties of the resource instance. Fields that do not mention RIS are constant across all resource instances.

Access to MPAMF_MBWUMON_IDR is not affected by MSMON_CFG_MON_SEL.RIS.

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.

MPAMF_MBWUMON_IDR can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rt	0x0090	MPAMF_MBWUMON_IDR_rt

This interface is accessible as follows:

- When FEAT_RME is implemented accesses to this register are RO.

MPAMF_MBWUMON_IDR can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rl	0x0090	MPAMF_MBWUMON_IDR_rl

This interface is accessible as follows:

- When FEAT_RME is implemented accesses to this register are RO.

11.3.10 MPAMF_MSMON_IDR, MPAM Resource Monitoring Identification Register

The MPAMF_MSMON_IDR characteristics are:

Purpose

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. MPAMF_MSMON_IDR_rt indicates Root monitoring features. MPAMF_MSMON_IDR_rl indicates Realm monitoring features.

If MPAMF_IDR.HAS_RIS is 1, fields that mention RIS must reflect the properties of the resource instance currently selected by MPAMCFG_PART_SEL.RIS. Fields that do not mention RIS are constant across all resource instances.

Configurations

The power domain of MPAMF_MSMON_IDR is IMPLEMENTATION DEFINED.

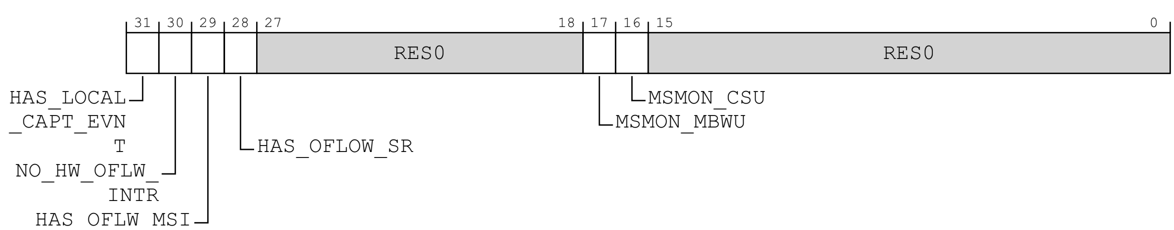
This register is present only when FEAT_MPAM is implemented and MPAMF_IDR.HAS_MSMON == 1. Otherwise, direct accesses to MPAMF_MSMON_IDR are RES0.

The power and reset domain of each MSC component is specific to that component.

Attributes

MPAMF_MSMON_IDR is a 32-bit register.

Field descriptions



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.

NO_HW_OFLW_INTR, bit [30]

When FEAT_MPAMv1p1 is implemented:

Does not have hardwired MPAM monitor overflow interrupt.

`0b0` Supports generating a hardwired interrupt to signal MPAM monitor overflow.

`0b1` No support for a hardwired interrupt to signal MPAM monitor overflow.

If this field is 0, the MSC supports generating a hardwired interrupt for monitor overflow events.

If this field is 0 and the HAS_OFLW_MSI field in this register is 1, the MSC supports generating both hardwired interrupts and MSI writes to signal interrupts.

Otherwise:

Reserved, RES0.

HAS_OFLW_MSI, bit [29]

When FEAT_MPAMv1p1 is implemented:

Has support for MSI writes to signal MPAM monitor overflow interrupts. These registers are implemented: MSMON_OFLOW_MSI_ADDR_L, MSMON_OFLOW_MSI_ADDR_H, MSMON_OFLOW_MSI_ATTR, MSMON_OFLOW_MSI_DATA and MSMON_OFLOW_MSI_MPAM.

`0b0` MSMON_OFLOW_MSI_ADDR_L, MSMON_OFLOW_MSI_ADDR_H, MSMON_OFLOW_MSI_ATTR, MSMON_OFLOW_MSI_DATA and MSMON_OFLOW_MSI_MPAM registers are not implemented.

`0b1` MSMON_OFLOW_MSI_ADDR_L, MSMON_OFLOW_MSI_ADDR_H, MSMON_OFLOW_MSI_ATTR, MSMON_OFLOW_MSI_DATA and MSMON_OFLOW_MSI_ATTR are implemented and can be used to generate writes to signal MPAM monitor overflow interrupts.

If MPAMF_MSMON_IDR.NO_HW_OFLW_INTR is 1 and this bit is 0, this MSC does not support monitor overflow interrupts.

Otherwise:

Reserved, RES0.

HAS_OFLOW_SR, bit [28]

When FEAT_MPAMv1p1 is implemented:

Has MPAM monitor overflow status register MSMON_OFLOW_SR.

`0b0` Does not have MSMON_OFLOW_SR.

`0b1` Supports MSMON_OFLOW_SR.

Otherwise:

Reserved, RES0.

Bits [27:18]

Reserved, RES0.

MSMON_MBWU, bit [17]

Memory bandwidth usage monitoring. Indicates whether MPAM monitoring for Memory Bandwidth Usage by PARTID and PMG is implemented and whether the following bandwidth usage registers are accessible:

- MPAMF_MBWUMON_IDR, MSMON_CFG_MBWU_CTL, MSMON_CFG_MBWU_FLT, MSMON_MBWU.
- The optional MSMON_MBWU_CAPTURE.
- If MPAM v0.1 or MPAM v1.1 is implemented, the optional MSMON_MBWU_L and the optional MSMON_MBWU_L_CAPTURE.

`0b0` Does not have monitoring for memory bandwidth usage and does not use the bandwidth usage registers.

`0b1` Has monitoring of memory bandwidth usage and uses the bandwidth usage registers.

If RIS is implemented, this field indicates that memory bandwidth usage monitoring is implemented for the resource instance selected by MPAMCFG_PART_SEL.RIS as described in MPAMF_MBWUMON_IDR.

MSMON_CSU, bit [16]

Cache storage usage monitoring. Indicates whether MPAM monitoring of cache storage usage by PARTID and PMG is implemented and the following registers are accessible:

- MPAMF_CSUMON_IDR, MSMON_CFG_CSU_CTL, MSMON_CFG_CSU_FLT, MSMON_CSU.

- The optional MSMON_CSU_CAPTURE.

`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.

If RIS is implemented, this field indicates that cache storage usage monitoring is implemented for the resource instance selected by MPAMCFG_PART_SEL.RIS as described in MPAMF_CSUMON_IDR.

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, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps.

MPAMF_MSMON_IDR is read-only.

MPAMF_MSMON_IDR must be readable from the Non-secure, Secure, Root, and Realm MPAM feature pages.

MPAMF_MSMON_IDR is permitted to have the same contents when read from the Secure, Non-secure, Root, and Realm MPAM feature pages unless the register contents are different for the different versions:

- MPAMF_MSMON_IDR_s is permitted to have either the same or different contents to MPAMF_MSMON_IDR_ns, MPAMF_MSMON_IDR_rt, or MPAMF_MSMON_IDR_rl.
- MPAMF_MSMON_IDR_ns is permitted to have either the same or different contents to MPAMF_MSMON_IDR_rt or MPAMF_MSMON_IDR_rl.
- MPAMF_MSMON_IDR_rt is permitted to have either the same or different contents to MPAMF_MSMON_IDR_rl.

There must be separate registers in the Secure (MPAMF_MSMON_IDR_s), Non-secure (MPAMF_MSMON_IDR_ns), Root (MPAMF_MSMON_IDR_rt), and Realm (MPAMF_MSMON_IDR_rl) MPAM feature pages.

When MPAMF_IDR.HAS_RIS is 1, MPAMF_MSMON_IDR shows the configuration of memory system monitoring for the resource instance selected by MPAMCFG_PART_SEL.RIS. Fields that mention RIS in their field descriptions have values that track the implemented properties of the resource instance. Fields that do not mention RIS are constant across all resource instances.

Access to MPAMF_MSMON_IDR is not affected by MSMON_CFG_MON_SEL.RIS.

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.

MPAMF_MSMON_IDR can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rt	0x0080	MPAMF_MSMON_IDR_rt

This interface is accessible as follows:

- When FEAT_RME is implemented accesses to this register are RO.

MPAMF_MSMON_IDR can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rl	0x0080	MPAMF_MSMON_IDR_rl

This interface is accessible as follows:

- When FEAT_RME is implemented 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

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.

When FEAT_RME is implemented: MPAMF_PARTID_NRW_rt indicates the largest Root internal PARTID. MPAMF_PARTID_NRW_rl indicates the largest Realm internal PARTID.

PARTID narrowing is global to the MSC and does not vary by resource instance.

Configurations

The power domain of MPAMF_PARTID_NRW_IDR is IMPLEMENTATION DEFINED.

This register is present only when FEAT_MPAM is implemented and MPAMF_IDR.HAS_PARTID_NRW == 1. Otherwise, direct accesses to MPAMF_PARTID_NRW_IDR are RES0.

The power and reset domain of each MSC component is specific to that component.

Attributes

MPAMF_PARTID_NRW_IDR is a 32-bit register.

Field descriptions

31 … 16	15 … 0
RES0	INTPARTID_MAX

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, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps.

MPAMF_PARTID_NRW_IDR is read-only.

MPAMF_PARTID_NRW_IDR must be readable from the Non-secure, Secure, Root, and Realm MPAM feature pages.

MPAMF_PARTID_NRW_IDR is permitted to have the same contents when read from the Secure, Non-secure, Root, and Realm MPAM feature pages unless the register contents are different for the different versions:

- MPAMF_PARTID_NRW_IDR_s is permitted to have either the same or different contents to MPAMF_PARTID_NRW_IDR_ns, MPAMF_PARTID_NRW_IDR_rt, or MPAMF_PARTID_NRW_IDR_rl.
- MPAMF_PARTID_NRW_IDR_ns is permitted to have either the same or different contents to MPAMF_PARTID_NRW_IDR_rt or MPAMF_PARTID_NRW_IDR_rl.
- MPAMF_PARTID_NRW_IDR_rt is permitted to have either the same or different contents to MPAMF_PARTID_NRW_IDR_rl.

There must be separate registers in the Secure (MPAMF_PARTID_NRW_IDR_s), Non-secure (MPAMF_PARTID_NRW_IDR_ns), Root (MPAMF_PARTID_NRW_IDR_rt), and Realm (MPAMF_PARTID_NRW_IDR_rl) 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.

MPAMF_PARTID_NRW_IDR can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rt	0x0050	MPAMF_PARTID_NRW_IDR_rt

This interface is accessible as follows:

- When FEAT_RME is implemented 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_rl	0x0050	MPAMF_PARTID_NRW_IDR_rl

This interface is accessible as follows:

- When FEAT_RME is implemented accesses to this register are RO.

11.3.12 MPAMF_PRI_IDR, MPAM Priority Partitioning Identification Register

The MPAMF_PRI_IDR characteristics are:

Purpose

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. MPAMF_PRI_IDR_rt indicates priority partitioning features accessed from the Root MPAM feature page. MPAMF_PRI_IDR_rl indicates priority partitioning features accessed from the Realm MPAM feature page.

When MPAMF_IDR.HAS_RIS is 1, some fields in this register give information for the resource instance selected by MPAMCFG_PART_SEL.RIS. The description of every field that is affected by MPAMCFG_PART_SEL.RIS has that information within the field description.

Configurations

The power domain of MPAMF_PRI_IDR is IMPLEMENTATION DEFINED.

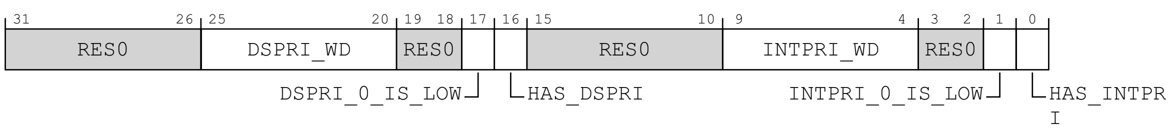
This register is present only when FEAT_MPAM is implemented and MPAMF_IDR.HAS_PRI_PART == 1. Otherwise, direct accesses to MPAMF_PRI_IDR are RES0.

The power and reset domain of each MSC component is specific to that component.

Attributes

MPAMF_PRI_IDR is a 32-bit register.

Field descriptions



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 16, inclusive.

If HAS_DSPRI == 0, this field must be 0.

If RIS is implemented, this field indicates the number of downstream priority bits for the resource instance selected by MPAMCFG_PART_SEL.RIS.

Bits [19:18]

Reserved, RES0.

DSPRI_0_IS_LOW, bit [17]

Indicates whether 0 in MPAMCFG_PRI.DSPRI is the lowest or the highest downstream 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.

If RIS is implemented, this field indicates that 0 is the lowest downstream priority for the resource instance selected by MPAMCFG_PART_SEL.RIS.

HAS_DSPRI, bit [16]

Indicates that the MPAMCFG_PRI register implements the DSPRI field.

`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.

If RIS is implemented, this field indicates that downstream priority is implemented for the resource instance selected by MPAMCFG_PART_SEL.RIS.

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 16, inclusive.

If HAS_INTPRI == 0, this field must be 0.

If RIS is implemented, this field indicates the number of internal priority bits for the resource instance selected by MPAMCFG_PART_SEL.RIS.

Bits [3:2]

Reserved, RES0.

INTPRI_0_IS_LOW, bit [1]

Indicates whether 0 in MPAMCFG_PRI.INTPRI is the lowest or the highest internal 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.

If RIS is implemented, this field indicates that 0 is the lowest internal priority for the resource instance selected by MPAMCFG_PART_SEL.RIS.

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.

If RIS is implemented, this field indicates that internal priority is implemented for the resource instance selected by MPAMCFG_PART_SEL.RIS.

Accessing the MPAMF_PRI_IDR:

This register is within the MPAM feature page memory frames. In a system that supports Secure, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps.

MPAMF_PRI_IDR is read-only.

MPAMF_PRI_IDR must be readable from the Non-secure, Secure, Root, and Realm MPAM feature pages.

MPAMF_PRI_IDR is permitted to have the same contents when read from the Secure, Non-secure, Root, and Realm MPAM feature pages unless the register contents are different for the different versions:

- MPAMF_PRI_IDR_s is permitted to have either the same or different contents to MPAMF_PRI_IDR_ns, MPAMF_PRI_IDR_rt, or MPAMF_PRI_IDR_rl.
- MPAMF_PRI_IDR_ns is permitted to have either the same or different contents to MPAMF_PRI_IDR_rt or MPAMF_PRI_IDR_rl.

- MPAMF_PRI_IDR_rt is permitted to have either the same or different contents to MPAMF_PRI_IDR_rl.

There must be separate registers in the Secure (MPAMF_PRI_IDR_s), Non-secure (MPAMF_PRI_IDR_ns), Root (MPAMF_PRI_IDR_rt), and Realm (MPAMF_PRI_IDR_rl) MPAM feature pages.

When MPAMF_IDR.HAS_RIS is 1, MPAMF_PRI_IDR shows the configuration of priority partitioning for the resource instance selected by MPAMCFG_PART_SEL.RIS. Fields that mention RIS in their field descriptions have values that track the implemented properties of the resource instance. Fields that do not mention RIS are constant across all resource instances.

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.

MPAMF_PRI_IDR can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rt	`0x0048`	MPAMF_PRI_IDR_rt

This interface is accessible as follows:

- When FEAT_RME is implemented accesses to this register are RO.

MPAMF_PRI_IDR can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rl	`0x0048`	MPAMF_PRI_IDR_rl

This interface is accessible as follows:

- When FEAT_RME is implemented 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.

This register is present only when FEAT_MPAM is implemented. Otherwise, direct accesses to MPAMF_SIDR are RES0.

The power and reset domain of each MSC component is specific to that component.

Attributes

MPAMF_SIDR is a 32-bit register.

Field descriptions

31 24	23 16	15 0
RES0	S_PMG_MAX	S_PARTID_MAX

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 only within the Secure MPAM feature page memory frame.

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 the 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. MPAMCFG_CMAX_rt controls the cache maximum capacity for the Root PARTID selected by the Root instance of MPAMCFG_PART_SEL. MPAMCFG_CMAX_rl controls the cache maximum capacity for the Realm PARTID selected by the Realm instance of MPAMCFG_PART_SEL.

If MPAMF_IDR.HAS_RIS is 1, the control settings accessed are those of the resource instance currently selected by MPAMCFG_PART_SEL.RIS and the PARTID selected by MPAMCFG_PART_SEL.PARTID_SEL.

Configurations

The power domain of MPAMCFG_CMAX is IMPLEMENTATION DEFINED.

This register is present only when FEAT_MPAM is implemented and MPAMF_IDR.HAS_CCAP_PART == 1. Otherwise, direct accesses to MPAMCFG_CMAX are RES0.

The power and reset domain of each MSC component is specific to that component.

Attributes

MPAMCFG_CMAX is a 32-bit register.

Field descriptions

31 – 16	15 – 0
RES0	CMAX

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 1.0 minus one half to the power w.

Accessing the MPAMCFG_CMAX:

This register is within the MPAM feature page memory frames.

In a system that supports Secure, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps.

- 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_rt must be accessible from the Root MPAM feature page.
- MPAMCFG_CMAX_rl must be accessible from the Realm MPAM feature page.

MPAMCFG_CMAX_s, MPAMCFG_CMAX_ns, MPAMCFG_CMAX_rt, and MPAMCFG_CMAX_rl must be separate registers.

- The Secure instance (MPAMCFG_CMAX_s) accesses the cache capacity partitioning used for Secure PARTIDs.
- The Non-secure instance (MPAMCFG_CMAX_ns) accesses the cache capacity partitioning used for Non-secure PARTIDs.
- The Root instance (MPAMCFG_CMAX_rt) accesses the cache capacity partitioning used for Root PARTIDs.
- The Realm instance (MPAMCFG_CMAX_rl) accesses the cache capacity partitioning used for Realm PARTIDs.

When RIS is implemented, loads and stores to MPAMCFG_CMAX access the cache maximum capacity partitioning configuration settings for the cache resource instance selected by MPAMCFG_PART_SEL.RIS and the PARTID selected by MPAMCFG_PART_SEL.PARTID_SEL.

When RIS is not implemented, loads and stores to MPAMCFG_CMAX access the cache maximum capacity partitioning configuration settings for the PARTID selected by MPAMCFG_PART_SEL.PARTID_SEL.

When PARTID narrowing is implemented, loads and stores to MPAMCFG_CMAX access the cache maximum capacity partitioning configuration settings for the internal PARTID selected by MPAMCFG_PART_SEL.PARTID_SEL, and MPAMCFG_PART_SEL.INTERNAL must be 1.

When PARTID narrowing is not implemented, loads and stores to MPAMCFG_CMAX access the cache maximum capacity partitioning configuration settings for the request PARTID selected by MPAMCFG_PART_SEL.PARTID_SEL, and MPAMCFG_PART_SEL.INTERNAL must be 0.

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.

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_rt	0x0108	MPAMCFG_CMAX_rt

This interface is accessible as follows:

- When FEAT_RME is implemented accesses to this register are RW.

MPAMCFG_CMAX can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rl	0x0108	MPAMCFG_CMAX_rl

This interface is accessible as follows:

- When FEAT_RME is implemented accesses to this register are RW.

11.4.2 MPAMCFG_CPBM<n>, MPAM Cache Portion Bitmap Partition Configuration Register, n = 0 - 1023

The MPAMCFG_CPBM<n> characteristics are:

Purpose

The MPAMCFG_CPBM<n> register array gives access to the cache portion bitmap. Each register in the array is a read/write register that configures the cache portions numbered from <n * 32> to <31 + (n * 32)> that a PARTID is allowed to allocate.

After setting MPAMCFG_PART_SEL with a PARTID, software writes to the MPAMCFG_CPBM<n> register to configure which cache portions the PARTID is allowed to allocate.

The MPAMCFG_CPBM<n> register that contains the bitmap bit corresponding to cache portion p has n equal to p[15:5]. The field, P<x>, of that MPAMCFG_CPBM<n> register that contains the bitmap bit corresponding to cache portion p has x equal to p[4:0].

MPAMCFG_CPBM<n>_s controls cache portions for the Secure PARTID selected by the Secure instance of MPAMCFG_PART_SEL. MPAMCFG_CPBM<n>_ns controls the cache portions for the Non-secure PARTID selected by the Non-secure instance of MPAMCFG_PART_SEL. MPAMCFG_CPBM<n>_rt controls cache portions for the Root PARTID selected by the Root instance of MPAMCFG_PART_SEL. MPAMCFG_CPBM<n>_rl controls the cache portions for the Realm PARTID selected by the Non-secure instance of MPAMCFG_PART_SEL.

If MPAMF_IDR.HAS_RIS is 1, the control settings accessed are those of the resource instance currently selected by MPAMCFG_PART_SEL.RIS and the PARTID selected by MPAMCFG_PART_SEL.PARTID_SEL.

Configurations

The power domain of MPAMCFG_CPBM<n> is IMPLEMENTATION DEFINED.

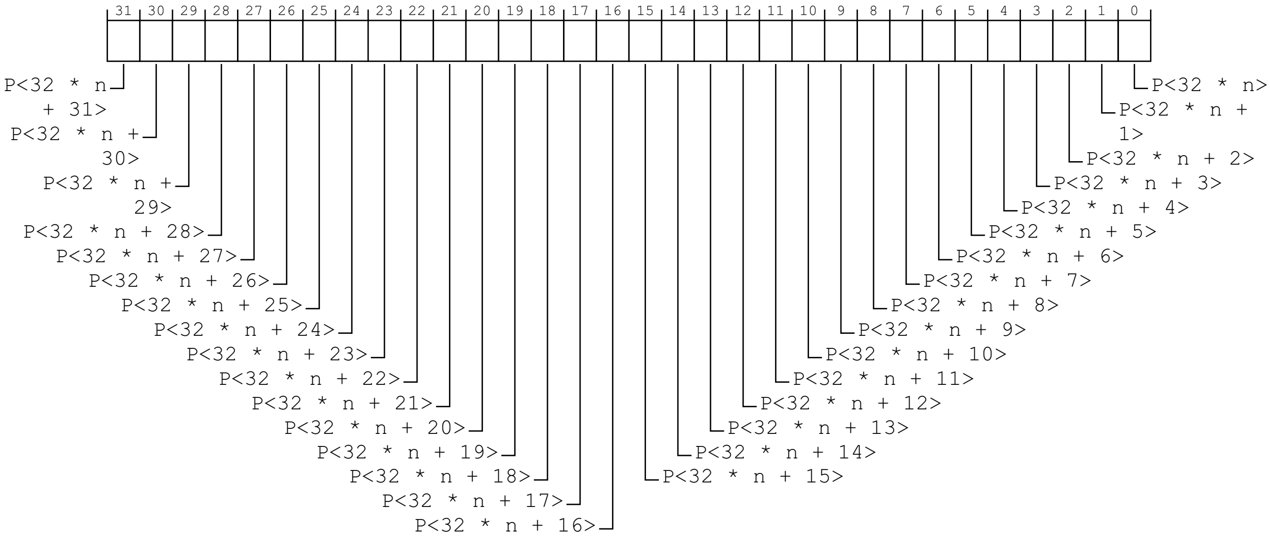
This register is present only when FEAT_MPAM is implemented and MPAMF_IDR.HAS_CPOR_PART == 1. Otherwise, direct accesses to MPAMCFG_CPBM<n> are RES0.

The power and reset domain of each MSC component is specific to that component.

Attributes

MPAMCFG_CPBM<n> is a 32-bit register.

Field descriptions



P<x + (n * 32)>, bit [x], for x = 31 to 0

Portion allocation control bit. Each cache portion allocation control bit, MPAMCFG_CPBM<n>.P<x>, grants permission to the PARTID selected by MPAMCFG_PART_SEL to allocate cache lines within cache portion <x + (n * 32)>.

`0b0` The PARTID is not permitted to allocate into cache portion <x + (n * 32)>.

`0b1` The PARTID is permitted to allocate within cache portion <x + (n * 32)>.

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 an array of 32-bit MPAMCFG_CPBM<n> registers to access the cache portion bitmap, up to 1024 registers.

Bits MPAMCFG_CPBM<n>.P<<x + (n * 32)>>, where <x + (n * 32)> is greater than or equal to CPBM_WD, are RES0:

- If n > MPAMF_CPOR_IDR.CPBM_WD[15:5], the entire 32 P<x> are RES0.
- If n == MPAMF_CPOR_IDR.CPBM_WD[15:5], bits [31: CPBM_WD[4:0]] are RES0 and the remaining bits are valid.
- If n < MPAMF_CPOR_IDR.CPBM_WD[15:5], the entire 32 P<x> are valid.

Accessing the MPAMCFG_CPBM<n>:

This register is within the MPAM feature page memory frames.

In a system that supports Secure, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps.

- MPAMCFG_CPBM<n>_s must be accessible from the Secure MPAM feature page.
- MPAMCFG_CPBM<n>_ns must be accessible from the Non-secure MPAM feature page.
- MPAMCFG_CPBM<n>_rt must be accessible from the Root MPAM feature page.
- MPAMCFG_CPBM<n>_rl must be accessible from the Realm MPAM feature page.

MPAMCFG_CPBM<n>_s, MPAMCFG_CPBM<n>_ns, MPAMCFG_CPBM<n>_rt, and MPAMCFG_CPBM<n>_rl must be separate registers.

- The Secure instance (MPAMCFG_CPBM<n>_s) accesses the cache portion bitmap used for Secure PARTIDs.
- The Non-secure instance (MPAMCFG_CPBM<n>_ns) accesses the cache portion bitmap used for Non-secure PARTIDs.
- The Root instance (MPAMCFG_CPBM<n>_rt) accesses the cache portion bitmap used for Root PARTIDs.
- The Realm instance (MPAMCFG_CPBM<n>_rl) accesses the cache portion bitmap used for Realm PARTIDs.

When RIS is implemented, loads and stores to MPAMCFG_CPBM<n> access the cache portion bitmap configuration settings for the cache resource instance selected by MPAMCFG_PART_SEL.RIS and the PARTID selected by MPAMCFG_PART_SEL.PARTID_SEL.

When RIS is not implemented, loads and stores to MPAMCFG_CPBM<n> access the cache portion bitmap configuration settings for the PARTID selected by MPAMCFG_PART_SEL.PARTID_SEL.

When PARTID narrowing is implemented, loads and stores to MPAMCFG_CPBM<n> access the cache portion bitmap configuration settings for the internal PARTID selected by MPAMCFG_PART_SEL.PARTID_SEL, and MPAMCFG_PART_SEL.INTERNAL must be 1.

When PARTID narrowing is not implemented, loads and stores to MPAMCFG_CPBM<n> access the cache portion bitmap configuration settings for the request PARTID selected by MPAMCFG_PART_SEL.PARTID_SEL, and MPAMCFG_PART_SEL.INTERNAL must be 0.

MPAMCFG_CPBM<n> can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_s	`0x1000` + (4 * n)	MPAMCFG_CPBM<n>_s

This interface is accessible as follows:

- Accesses to this register are RW.

MPAMCFG_CPBM<n> can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_ns	`0x1000` + (4 * n)	MPAMCFG_CPBM<n>_ns

This interface is accessible as follows:

- Accesses to this register are RW.

MPAMCFG_CPBM<n> can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rt	`0x1000` + (4 * n)	MPAMCFG_CPBM<n>_rt

This interface is accessible as follows:

- When FEAT_RME is implemented accesses to this register are RW.

MPAMCFG_CPBM<n> can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rl	0x1000 + (4 * n)	MPAMCFG_CPBM<n>_rl

This interface is accessible as follows:

- When FEAT_RME is implemented 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. MPAMCFG_INTPARTID_rt controls the mapping for the Root PARTID selected by the Root instance of MPAMCFG_PART_SEL. MPAMCFG_INTPARTID_rl controls the mapping for the Realm PARTID selected by the Realm 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.

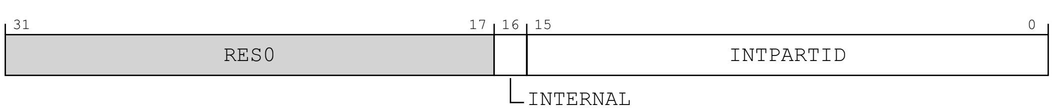
This register is present only when FEAT_MPAM is implemented and MPAMF_IDR.HAS_PARTID_NRW == 1. Otherwise, direct accesses to MPAMCFG_INTPARTID are RES0.

The power and reset domain of each MSC component is specific to that component.

Attributes

MPAMCFG_INTPARTID is a 32-bit register.

Field descriptions



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, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps.

- 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_rt must be accessible from the Root MPAM feature page.
- MPAMCFG_INTPARTID_rl must be accessible from the Realm MPAM feature page.

MPAMCFG_INTPARTID_s, MPAMCFG_INTPARTID_ns, MPAMCFG_INTPARTID_rt, and MPAMCFG_INTPARTID_rl must be separate registers.

- The Secure instance (MPAMCFG_INTPARTID_s) accesses the PARTID narrowing used for Secure PARTIDs.
- The Non-secure instance (MPAMCFG_INTPARTID_ns) accesses the PARTID narrowing used for Non-secure PARTIDs.
- The Root instance (MPAMCFG_INTPARTID_rt) accesses the PARTID narrowing used for Root PARTIDs.
- The Realm instance (MPAMCFG_INTPARTID_rl) accesses the PARTID narrowing used for Realm PARTIDs.

When RIS is implemented, loads and stores to MPAMCFG_INTPARTID access the PARTID narrowing configuration settings without being affected by MPAMCFG_PART_SEL.RIS.

Loads and stores to MPAMCFG_INTPARTID access the PARTID narrowing configuration settings for the request PARTID selected by MPAMCFG_PART_SEL.PARTID_SEL, and MPAMCFG_PART_SEL.INTERNAL must be 0.

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.

MPAMCFG_INTPARTID can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rt	`0x0600`	MPAMCFG_INTPARTID_rt

This interface is accessible as follows:

- When FEAT_RME is implemented accesses to this register are RW.

MPAMCFG_INTPARTID can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rl	`0x0600`	MPAMCFG_INTPARTID_rl

This interface is accessible as follows:

- When FEAT_RME is implemented 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. MPAMCFG_MBW_MAX_rt controls the maximum bandwidth for the Root PARTID selected by the Root instance of MPAMCFG_PART_SEL. MPAMCFG_MBW_MAX_rl controls the maximum bandwidth for the Realm PARTID selected by the Realm 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.

If MPAMF_IDR.HAS_RIS is 1, the control settings accessed are those of the resource instance currently selected by MPAMCFG_PART_SEL.RIS and the PARTID selected by MPAMCFG_PART_SEL.PARTID_SEL.

Configurations

The power domain of MPAMCFG_MBW_MAX is IMPLEMENTATION DEFINED.

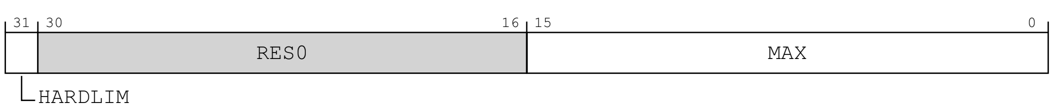
This register is present only when FEAT_MPAM is implemented, MPAMF_IDR.HAS_MBW_PART == 1 and MPAMF_MBW_IDR.HAS_MAX == 1. Otherwise, direct accesses to MPAMCFG_MBW_MAX are RES0.

The power and reset domain of each MSC component is specific to that component.

Attributes

MPAMCFG_MBW_MAX is a 32-bit register.

Field descriptions



HARDLIM, bit [31]

Hard bandwidth limiting.

`0b0` When MAX bandwidth is exceeded, the partition contends with a low preference for downstream bandwidth beyond MAX.

`0b1` When MAX bandwidth is exceeded, the partition does not use any more bandwidth until the memory bandwidth measurement for the partition falls below MAX.

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 bandwidth that can be represented in an implementation with w implemented bits is 1.0 minus one half to the power w.

Accessing the MPAMCFG_MBW_MAX:

This register is within the MPAM feature page memory frames.

In a system that supports Secure, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps.

- 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_rt must be accessible from the Root MPAM feature page.
- MPAMCFG_MBW_MAX_rl must be accessible from the Realm MPAM feature page.

MPAMCFG_MBW_MAX_s, MPAMCFG_MBW_MAX_ns, MPAMCFG_MBW_MAX_rt, and MPAMCFG_MBW_MAX_rl must be separate registers.

- The Secure instance (MPAMCFG_MBW_MAX_s) accesses the memory maximum bandwidth partitioning used for Secure PARTIDs.
- The Non-secure instance (MPAMCFG_MBW_MAX_ns) accesses the memory maximum bandwidth partitioning used for Non-secure PARTIDs.
- The Root instance (MPAMCFG_MBW_MAX_rt) accesses the memory maximum bandwidth partitioning used for Root PARTIDs.
- The Realm instance (MPAMCFG_MBW_MAX_rl) accesses the memory maximum bandwidth partitioning used for Realm PARTIDs.

When RIS is implemented, loads and stores to MPAMCFG_MBW_MAX access the memory maximum bandwidth partitioning configuration settings for the bandwidth resource instance selected by MPAMCFG_PART_SEL.RIS and the PARTID selected by MPAMCFG_PART_SEL.PARTID_SEL.

When RIS is not implemented, loads and stores to MPAMCFG_MBW_MAX access the memory maximum bandwidth partitioning configuration settings for the PARTID selected by MPAMCFG_PART_SEL.PARTID_SEL.

When PARTID narrowing is implemented, loads and stores to MPAMCFG_MBW_MAX access the memory maximum bandwidth partitioning configuration settings for the internal PARTID selected by MPAMCFG_PART_SEL.PARTID_SEL, and MPAMCFG_PART_SEL.INTERNAL must be 1.

When PARTID narrowing is not implemented, loads and stores to MPAMCFG_MBW_MAX access the memory maximum bandwidth partitioning configuration settings for the request PARTID selected by MPAMCFG_PART_SEL.PARTID_SEL, and MPAMCFG_PART_SEL.INTERNAL must be 0.

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.

MPAMCFG_MBW_MAX can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rt	0x0208	MPAMCFG_MBW_MAX_rt

This interface is accessible as follows:

- When FEAT_RME is implemented accesses to this register are RW.

MPAMCFG_MBW_MAX can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rl	0x0208	MPAMCFG_MBW_MAX_rl

This interface is accessible as follows:

- When FEAT_RME is implemented accesses to this register are RW.

11.4.5 MPAMCFG_MBW_MIN, MPAM Memory Bandwidth Minimum 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. MPAMCFG_MBW_MIN_rt controls the minimum bandwidth for the Root PARTID selected by the Root instance of MPAMCFG_PART_SEL. MPAMCFG_MBW_MIN_rl controls the minimum bandwidth for the Realm PARTID selected by the Realm instance of MPAMCFG_PART_SEL.

A PARTID that has used less than MIN is given preferential access to bandwidth.

If MPAMF_IDR.HAS_RIS is 1, the control settings accessed are those of the resource instance currently selected by MPAMCFG_PART_SEL.RIS and the PARTID selected by MPAMCFG_PART_SEL.PARTID_SEL.

Configurations

The power domain of MPAMCFG_MBW_MIN is IMPLEMENTATION DEFINED.

This register is present only when FEAT_MPAM is implemented, MPAMF_IDR.HAS_MBW_PART == 1 and MPAMF_MBW_IDR.HAS_MIN == 1. Otherwise, direct accesses to MPAMCFG_MBW_MIN are RES0.

The power and reset domain of each MSC component is specific to that component.

Attributes

MPAMCFG_MBW_MIN is a 32-bit register.

Field descriptions

31 – 16	15 – 0
RES0	MIN

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 bandwidth that can be represented in an implementation with w implemented bits is 1.0 minus one half to the power w.

Accessing the MPAMCFG_MBW_MIN:

This register is within the MPAM feature page memory frames.

In a system that supports Secure, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps.

- 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_rt must be accessible from the Root MPAM feature page.
- MPAMCFG_MBW_MIN_rl must be accessible from the Realm MPAM feature page.

MPAMCFG_MBW_MIN_s, MPAMCFG_MBW_MIN_ns, MPAMCFG_MBW_MIN_rt, and MPAMCFG_MBW_MIN_rl must be separate registers.

- The Secure instance (MPAMCFG_MBW_MIN_s) accesses the memory minimum bandwidth partitioning used for Secure PARTIDs.
- The Non-secure instance (MPAMCFG_MBW_MIN_ns) accesses the memory minimum bandwidth partitioning used for Non-secure PARTIDs.
- The Root instance (MPAMCFG_MBW_MIN_rt) accesses the memory minimum bandwidth partitioning used for Root PARTIDs.
- The Realm instance (MPAMCFG_MBW_MIN_rl) accesses the memory minimum bandwidth partitioning used for Realm PARTIDs.

When RIS is implemented, loads and stores to MPAMCFG_MBW_MIN access the memory minimum bandwidth partitioning configuration settings for the bandwidth resource instance selected by MPAMCFG_PART_SEL.RIS and the PARTID selected by MPAMCFG_PART_SEL.PARTID_SEL.

When RIS is not implemented, loads and stores to MPAMCFG_MBW_MIN access the memory minimum bandwidth partitioning configuration settings for the PARTID selected by MPAMCFG_PART_SEL.PARTID_SEL.

When PARTID narrowing is implemented, loads and stores to MPAMCFG_MBW_MIN access the memory minimum bandwidth partitioning configuration settings for the internal PARTID selected by MPAMCFG_PART_SEL.PARTID_SEL, and MPAMCFG_PART_SEL.INTERNAL must be 1.

When PARTID narrowing is not implemented, loads and stores to MPAMCFG_MBW_MIN access the memory minimum bandwidth partitioning configuration settings for the request PARTID selected by MPAMCFG_PART_SEL.PARTID_SEL, and MPAMCFG_PART_SEL.INTERNAL must be 0.

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.

MPAMCFG_MBW_MIN can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rt	0x0200	MPAMCFG_MBW_MIN_rt

This interface is accessible as follows:

- When FEAT_RME is implemented accesses to this register are RW.

MPAMCFG_MBW_MIN can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rl	0x0200	MPAMCFG_MBW_MIN_rl

This interface is accessible as follows:

- When FEAT_RME is implemented accesses to this register are RW.

11.4.6 MPAMCFG_MBW_PBM<n>, MPAM Bandwidth Portion Bitmap Partition Configuration Register, n = 0 - 127

The MPAMCFG_MBW_PBM<n> characteristics are:

Purpose

The MPAMCFG_MBW_PBM<n> register array gives access to the memory bandwidth portion bitmap. Each register in the array is a read/write register that configures the bandwidth portions <32 * n> to <(32 * n) + 31> that a PARTID is allowed to allocate.

After setting MPAMCFG_PART_SEL with a PARTID, software writes to one or more of the MPAMCFG_MBW_PBM<n> registers to configure which bandwidth portions the PARTID is allowed to allocate.

The MPAMCFG_MBW_PBM<n> register that contains the bitmap bit corresponding to memory bandwidth portion p has n equal to p[11:5]. The field, P<<x + (32 * n)>> of that MPAMCFG_MBW_PBM<n> register that contains the bitmap bit corresponding to memory bandwidth portion p has x equal to p[4:0].

The MPAMCFG_MBW_PBM<n>_s registers control the bandwidth portion bitmap for the Secure PARTID selected by the Secure instance of MPAMCFG_PART_SEL. The MPAMCFG_MBW_PBM<n>_ns registers control the bandwidth portion bitmap for the Non-secure PARTID selected by the Non-secure instance of MPAMCFG_PART_SEL. The MPAMCFG_MBW_PBM<n>_rt registers control the bandwidth portion bitmap for the Root PARTID selected by the Root instance of MPAMCFG_PART_SEL. The MPAMCFG_MBW_PBM<n>_rl registers control the bandwidth portion bitmap for the Realm PARTID selected by the Realm instance of MPAMCFG_PART_SEL.

If MPAMF_IDR.HAS_RIS is 1, the control settings accessed are those of the resource instance currently selected by MPAMCFG_PART_SEL.RIS and the PARTID selected by MPAMCFG_PART_SEL.PARTID_SEL.

Configurations

The power domain of MPAMCFG_MBW_PBM<n> is IMPLEMENTATION DEFINED.

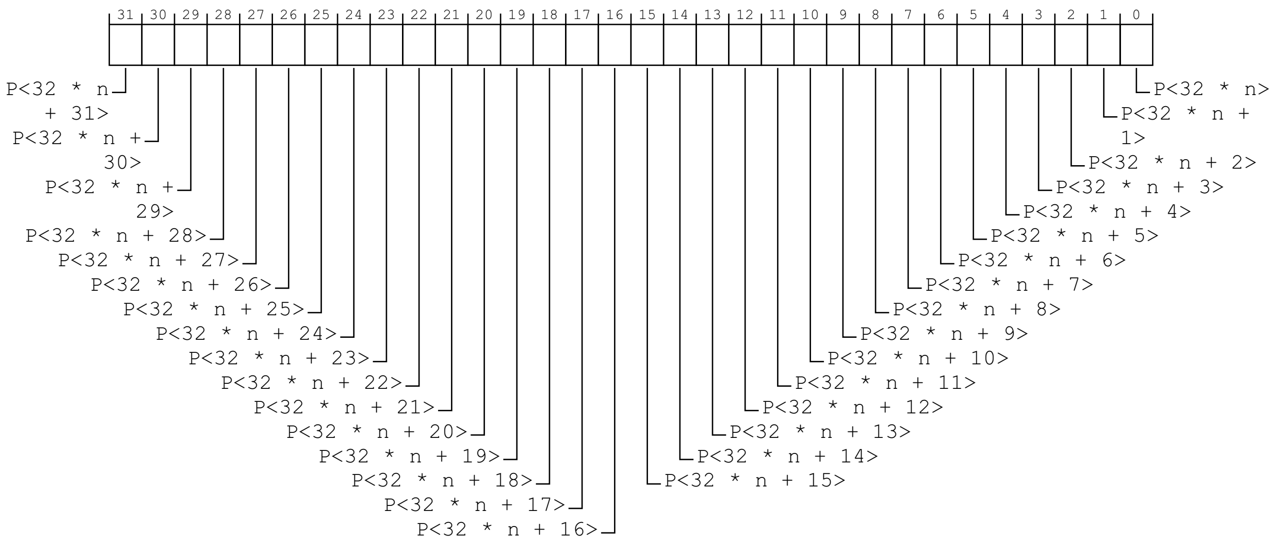
This register is present only when FEAT_MPAM is implemented, MPAMF_IDR.HAS_MBW_PART == 1 and MPAMF_MBW_IDR.HAS_PBM == 1. Otherwise, direct accesses to MPAMCFG_MBW_PBM<n> are RES0.

The power and reset domain of each MSC component is specific to that component.

Attributes

MPAMCFG_MBW_PBM<n> is a 32-bit register.

Field descriptions



P<x + (32 * n)>, bit [x], for x = 31 to 0

Portion allocation control bit. Each bandwidth portion allocation control bit MPAMCFG_MBW_PBM<n>.P<<x + (32 * n)>> grants permission to the PARTID selected by MPAMCFG_PART_SEL to allocate bandwidth within bandwidth portion <x + (32 * n)>.

`0b0` The PARTID is not permitted to allocate into bandwidth portion <x + (32 * n)>.

`0b1` The PARTID is permitted to allocate within bandwidth portion <x + (32 * 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 bandwidth portion bitmap, up to 128 32-bit registers.

Bits MPAMCFG_MBW_PBM<n>.P<<x + (32 * n)>>, where <x + (32 * n)>is greater than or equal to BWPBM_WD are RES0:

- If n > MPAMF_MBW_IDR.BWPBM_WD[11:5], the entire 32 P<x> are RES0.
- If n == MPAMF_MBW_IDR.BWPBM_WD[11:5], bits [31: BWPBM_WD[4:0]] are RES0 and the remaining bits are valid.
- If n < MPAMF_MBW_IDR.BWPBM_WD[11:5], the entire 32 P<x> are valid.

Accessing the MPAMCFG_MBW_PBM<n>:

This register is within the MPAM feature page memory frames.

In a system that supports Secure, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps.

- MPAMCFG_MBW_PBM<n>_s must be accessible from the Secure MPAM feature page.
- MPAMCFG_MBW_PBM<n>_ns must be accessible from the Non-secure MPAM feature page.
- MPAMCFG_MBW_PBM<n>_rt must be accessible from the Root MPAM feature page.
- MPAMCFG_MBW_PBM<n>_rl must be accessible from the Realm MPAM feature page.

MPAMCFG_MBW_PBM<n>_s, MPAMCFG_MBW_PBM<n>_ns, MPAMCFG_MBW_PBM<n>_rt, and MPAMCFG_MBW_PBM<n>_rl must be separate registers.

- The Secure instance (MPAMCFG_MBW_PBM<n>_s) accesses the memory bandwidth portion bitmap used for Secure PARTIDs.
- The Non-secure instance (MPAMCFG_MBW_PBM<n>_ns) accesses the memory bandwidth portion bitmap used for Non-secure PARTIDs.
- The Root instance (MPAMCFG_MBW_PBM<n>_rt) accesses the memory bandwidth portion bitmap used for Root PARTIDs.
- The Realm instance (MPAMCFG_MBW_PBM<n>_rl) accesses the memory bandwidth portion bitmap used for Realm PARTIDs.

When RIS is implemented, loads and stores to MPAMCFG_MBW_PBM<n> access the memory bandwidth portion bitmap configuration settings for the bandwidth resource instance selected by MPAMCFG_PART_SEL.RIS and the PARTID selected by MPAMCFG_PART_SEL.PARTID_SEL.

When RIS is not implemented, loads and stores to MPAMCFG_MBW_PBM<n> access the memory bandwidth portion bitmap configuration settings for the PARTID selected by MPAMCFG_PART_SEL.PARTID_SEL.

When PARTID narrowing is implemented, loads and stores to MPAMCFG_MBW_PBM<n> access the memory bandwidth portion bitmap configuration settings for the internal PARTID selected by MPAMCFG_PART_SEL.PARTID_SEL, and MPAMCFG_PART_SEL.INTERNAL must be 1.

When PARTID narrowing is not implemented, loads and stores to MPAMCFG_MBW_PBM<n> access the memory bandwidth portion bitmap configuration settings for the request PARTID selected by MPAMCFG_PART_SEL.PARTID_SEL, and MPAMCFG_PART_SEL.INTERNAL must be 0.

MPAMCFG_MBW_PBM<n> can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_s	`0x2000` + (4 * n)	MPAMCFG_MBW_PBM<n>_s

This interface is accessible as follows:

- Accesses to this register are RW.

MPAMCFG_MBW_PBM<n> can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_ns	`0x2000` + (4 * n)	MPAMCFG_MBW_PBM<n>_ns

This interface is accessible as follows:

- Accesses to this register are RW.

MPAMCFG_MBW_PBM<n> can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rt	`0x2000` + (4 * n)	MPAMCFG_MBW_PBM<n>_rt

This interface is accessible as follows:

- When FEAT_RME is implemented accesses to this register are RW.

MPAMCFG_MBW_PBM<n> can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rl	`0x2000` + (4 * n)	MPAMCFG_MBW_PBM<n>_rl

This interface is accessible as follows:

- When FEAT_RME is implemented 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

Controls the proportional stride of memory bandwidth that the PARTID selected by MPAMCFG_PART_SEL uses.

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. MPAMCFG_MBW_PROP_rt controls the bandwidth proportional stride for the Root PARTID selected by the Root instance of MPAMCFG_PART_SEL. MPAMCFG_MBW_PROP_rl controls the bandwidth proportional stride for the Realm PARTID selected by the Realm instance of MPAMCFG_PART_SEL.

Proportional stride is a relative cost of bandwidth requested by one PARTID in relation to the costs of the bandwidths requested by each other PARTID also competing to use the bandwidth.

If MPAMF_IDR.HAS_RIS is 1, the control settings accessed are those of the resource instance currently selected by MPAMCFG_PART_SEL.RIS and the PARTID selected by MPAMCFG_PART_SEL.PARTID_SEL.

Configurations

The power domain of MPAMCFG_MBW_PROP is IMPLEMENTATION DEFINED.

This register is present only when FEAT_MPAM is implemented, MPAMF_IDR.HAS_MBW_PART == 1 and MPAMF_MBW_IDR.HAS_PROP == 1. Otherwise, direct accesses to MPAMCFG_MBW_PROP are RES0.

The power and reset domain of each MSC component is specific to that component.

Attributes

MPAMCFG_MBW_PROP is a 32-bit register.

Field descriptions

31	30 – 16	15 – 0
EN	RES0	STRIDEM1

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, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps.

- 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_rt must be accessible from the Root MPAM feature page.
- MPAMCFG_MBW_PROP_rl must be accessible from the Realm MPAM feature page.

MPAMCFG_MBW_PROP_s, MPAMCFG_MBW_PROP_ns, MPAMCFG_MBW_PROP_rt, and MPAMCFG_MBW_PROP_rl must be separate registers.

- The Secure instance (MPAMCFG_MBW_PROP_s) accesses the memory proportional stride bandwidth partitioning used for Secure PARTIDs.
- The Non-secure instance (MPAMCFG_MBW_PROP_ns) accesses the memory proportional stride bandwidth partitioning used for Non-secure PARTIDs.
- The Root instance (MPAMCFG_MBW_PROP_rt) accesses the memory proportional stride bandwidth partitioning used for Root PARTIDs.
- The Realm instance (MPAMCFG_MBW_PROP_rl) accesses the memory proportional stride bandwidth partitioning used for Realm PARTIDs.

When RIS is implemented, loads and stores to MPAMCFG_MBW_PROP access the memory proportional stride bandwidth partitioning configuration settings for the bandwidth resource instance selected by MPAMCFG_PART_SEL.RIS and the PARTID selected by MPAMCFG_PART_SEL.PARTID_SEL.

When RIS is not implemented, loads and stores to MPAMCFG_MBW_PROP access the memory proportional stride bandwidth partitioning configuration settings for the PARTID selected by MPAMCFG_PART_SEL.PARTID_SEL.

When PARTID narrowing is implemented, loads and stores to MPAMCFG_MBW_PROP access the memory proportional stride bandwidth partitioning configuration settings for the internal PARTID selected by MPAMCFG_PART_SEL.PARTID_SEL, and MPAMCFG_PART_SEL.INTERNAL must be 1.

When PARTID narrowing is not implemented, loads and stores to MPAMCFG_MBW_PROP access the memory proportional stride bandwidth partitioning configuration settings for the request PARTID selected by MPAMCFG_PART_SEL.PARTID_SEL, and MPAMCFG_PART_SEL.INTERNAL must be 0.

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.

MPAMCFG_MBW_PROP can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rt	0x0500	MPAMCFG_MBW_PROP_rt

This interface is accessible as follows:

- When FEAT_RME is implemented accesses to this register are RW.

MPAMCFG_MBW_PROP can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rl	0x0500	MPAMCFG_MBW_PROP_rl

This interface is accessible as follows:

- When FEAT_RME is implemented 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 width for the Non-secure PARTID selected by the Non-secure instance of MPAMCFG_PART_SEL.
MPAMCFG_MBW_WINWD_rt reads and controls the bandwidth control window width for the Root PARTID selected by the Root instance of MPAMCFG_PART_SEL.
MPAMCFG_MBW_WINWD_rl reads and controls the bandwidth control window width for the Real PARTID selected by the Realm 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.

If MPAMF_IDR.HAS_RIS is 1, the control settings accessed are those of the resource instance currently selected by MPAMCFG_PART_SEL.RIS and the PARTID selected by MPAMCFG_PART_SEL.PARTID_SEL.

Configurations

The power domain of MPAMCFG_MBW_WINWD is IMPLEMENTATION DEFINED.

This register is present only when FEAT_MPAM is implemented and MPAMF_IDR.HAS_MBW_PART == 1. Otherwise, direct accesses to MPAMCFG_MBW_WINWD are RES0.

The power and reset domain of each MSC component is specific to that component.

Attributes

MPAMCFG_MBW_WINWD is a 32-bit register.

Field descriptions

31–24	23–8	7–0
RES0	US_INT	US_FRAC

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, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps.

- 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_rt must be accessible from the Root MPAM feature page.
- MPAMCFG_MBW_WINWD_rl must be accessible from the Realm MPAM feature page.

MPAMCFG_MBW_WINWD_s, MPAMCFG_MBW_WINWD_ns, MPAMCFG_MBW_WINWD_rt, and MPAMCFG_MBW_WINWD_rl must be separate registers.

- The Secure instance (MPAMCFG_MBW_WINWD_s) accesses the window width used for Secure PARTIDs.
- The Non-secure instance (MPAMCFG_MBW_WINWD_ns) accesses the window width used for Non-secure PARTIDs.
- The Root instance (MPAMCFG_MBW_WINWD_rt) accesses the window width used for Root PARTIDs.
- The Realm instance (MPAMCFG_MBW_WINWD_rl) accesses the window width used for Realm PARTIDs.

When RIS is implemented, loads and stores to MPAMCFG_MBW_WINWD access the window width configuration settings for the bandwidth resource instance selected by MPAMCFG_PART_SEL.RIS and the PARTID selected by MPAMCFG_PART_SEL.PARTID_SEL.

When RIS is not implemented, loads and stores to MPAMCFG_MBW_WINWD access the window width configuration settings for the PARTID selected by MPAMCFG_PART_SEL.PARTID_SEL.

When PARTID narrowing is implemented, loads and stores to MPAMCFG_MBW_WINWD access the window width configuration settings for the internal PARTID selected by MPAMCFG_PART_SEL.PARTID_SEL, and MPAMCFG_PART_SEL.INTERNAL must be 1.

When PARTID narrowing is not implemented, loads and stores to MPAMCFG_MBW_WINWD access the window width configuration settings for the request PARTID selected by MPAMCFG_PART_SEL.PARTID_SEL, and MPAMCFG_PART_SEL.INTERNAL must be 0.

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.WINDWR == 0 accesses to this register are RO.
- When MPAMF_MBW_IDR.WINDWR == 1 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.WINDWR == 0 accesses to this register are RO.
- When MPAMF_MBW_IDR.WINDWR == 1 accesses to this register are RW.

MPAMCFG_MBW_WINWD can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rt	0x0220	MPAMCFG_MBW_WINWD_rt

This interface is accessible as follows:

- When FEAT_RME is implemented and MPAMF_MBW_IDR.WINDWR == 0 accesses to this register are RO.
- When FEAT_RME is implemented and MPAMF_MBW_IDR.WINDWR == 1 accesses to this register are RW.

MPAMCFG_MBW_WINWD can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rl	0x0220	MPAMCFG_MBW_WINWD_rl

This interface is accessible as follows:

- When FEAT_RME is implemented and MPAMF_MBW_IDR.WINDWR == 0 accesses to this register are RO.
- When FEAT_RME is implemented and MPAMF_MBW_IDR.WINDWR == 1 accesses to this register are RW.

11.4.9 MPAMCFG_PART_SEL, MPAM Partition Configuration Selection Register

The MPAMCFG_PART_SEL characteristics are:

Purpose

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. MPAMCFG_PART_SEL_rt selects a Root PARTID to configure. MPAMCFG_PART_SEL_rl selects a Realm 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.

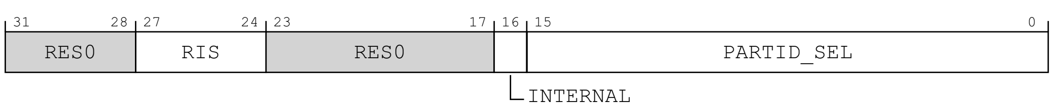
This register is present only when FEAT_MPAM is implemented and (MPAMF_IDR.HAS_CCAP_PART == 1, or MPAMF_IDR.HAS_CPOR_PART == 1, or MPAMF_IDR.HAS_MBW_PART == 1, or MPAMF_IDR.HAS_PRI_PART == 1, or MPAMF_IDR.HAS_PARTID_NRW == 1, or (MPAMF_IDR.EXT == 0 and MPAMF_IDR.HAS_IMPL_IDR == 1) or (MPAMF_IDR.EXT == 1, MPAMF_IDR.HAS_IMPL_IDR == 1 and MPAMF_IDR.NO_IMPL_PART == 0)). Otherwise, direct accesses to MPAMCFG_PART_SEL are RES0.

The power and reset domain of each MSC component is specific to that component.

Attributes

MPAMCFG_PART_SEL is a 32-bit register.

Field descriptions



Bits [31:28]

Reserved, RES0.

RIS, bits [27:24]

When (FEAT_MPAMv0p1 is implemented or FEAT_MPAMv1p1 is implemented), MPAMF_IDR.EXT == 1 and MPAMF_IDR.HAS_RIS == 1:

Resource Instance Selector. RIS selects one resource to configure through MPAMCFG registers and describe with MPAMF ID registers.

Otherwise:

Reserved, RES0.

Bits [23:17]

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 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.

MPAMCFG_PART_SEL can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rt	`0x0100`	MPAMCFG_PART_SEL_rt

This interface is accessible as follows:

- When FEAT_RME is implemented accesses to this register are RW.

MPAMCFG_PART_SEL can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rl	`0x0100`	MPAMCFG_PART_SEL_rl

This interface is accessible as follows:

- When FEAT_RME is implemented accesses to this register are RW.

11.4.10 MPAMCFG_PRI, MPAM Priority Partition Configuration Register

The MPAMCFG_PRI characteristics are:

Purpose

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. MPAMCFG_PRI_rt controls the priorities for the Root PARTID selected by the Root instance of MPAMCFG_PART_SEL. MPAMCFG_PRI_rl controls the priorities for the Realm PARTID selected by the Realm instance of MPAMCFG_PART_SEL.

If MPAMF_IDR.HAS_RIS is 1, the control settings accessed are those of the resource instance currently selected by MPAMCFG_PART_SEL.RIS and the PARTID selected by MPAMCFG_PART_SEL.PARTID_SEL.

Configurations

The power domain of MPAMCFG_PRI is IMPLEMENTATION DEFINED.

This register is present only when FEAT_MPAM is implemented and MPAMF_IDR.HAS_PRI_PART == 1. Otherwise, direct accesses to MPAMCFG_PRI are RES0.

The power and reset domain of each MSC component is specific to that component.

Attributes

MPAMCFG_PRI is a 32-bit register.

Field descriptions

31 … 16	15 … 0
DSPRI	INTPRI

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. If the implemented width is less than the width of this field, the least significant bits are used.

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 MPAMCFG_PART_SEL.

The implemented width of this field is MPAMF_PRI_IDR.INTPRI_WD bits. If the implemented width is less than the width of this field, the least significant bits are used.

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, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps.

- 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_rt must be accessible from the Root MPAM feature page.
- MPAMCFG_PRI_rl must be accessible from the Realm MPAM feature page.

MPAMCFG_PRI_s, MPAMCFG_PRI_ns, MPAMCFG_PRI_rt, and MPAMCFG_PRI_rl must be separate registers.

- The Secure instance (MPAMCFG_PRI_s) accesses the priority partitioning used for Secure PARTIDs.
- The Non-secure instance (MPAMCFG_PRI_ns) accesses the priority partitioning used for Non-secure PARTIDs.
- The Root instance (MPAMCFG_PRI_rt) accesses the priority partitioning used for Root PARTIDs.
- The Realm instance (MPAMCFG_PRI_rl) accesses the priority partitioning used for Realm PARTIDs.

When RIS is implemented, loads and stores to MPAMCFG_PRI access the priority partitioning configuration settings for the priority resource instance selected by MPAMCFG_PART_SEL.RIS and the PARTID selected by MPAMCFG_PART_SEL.PARTID_SEL.

When RIS is not implemented, loads and stores to MPAMCFG_PRI access the priority partitioning configuration settings for the PARTID selected by MPAMCFG_PART_SEL.PARTID_SEL.

When PARTID narrowing is implemented, loads and stores to MPAMCFG_PRI access the priority partitioning configuration settings for the internal PARTID selected by MPAMCFG_PART_SEL.PARTID_SEL, and MPAMCFG_PART_SEL.INTERNAL must be 1.

When PARTID narrowing is not implemented, loads and stores to MPAMCFG_PRI access the priority partitioning configuration settings for the request PARTID selected by MPAMCFG_PART_SEL.PARTID_SEL, and MPAMCFG_PART_SEL.INTERNAL must be 0.

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.

MPAMCFG_PRI can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rt	`0x0400`	MPAMCFG_PRI_rt

This interface is accessible as follows:

- When FEAT_RME is implemented accesses to this register are RW.

MPAMCFG_PRI can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rl	`0x0400`	MPAMCFG_PRI_rl

This interface is accessible as follows:

- When FEAT_RME is implemented 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

Generates a local capture event when written with bit[0] as 1.

MSMON_CAPT_EVNT_s generates local capture events for Secure monitor instances only or for Secure and Non-secure monitor instances. MSMON_CAPT_EVNT_ns generates local capture events for Non-secure monitor instances only. MSMON_CAPT_EVNT_rt generates local capture events for Root monitor instances only or for Root, Secure, Realm, and Non-secure monitor instances. MSMON_CAPT_EVNT_rl generates local capture events for Realm monitor instances or for for Realm monitor instances or Realm and Non-secure monitor instances.

Configurations

The power domain of MSMON_CAPT_EVNT is IMPLEMENTATION DEFINED.

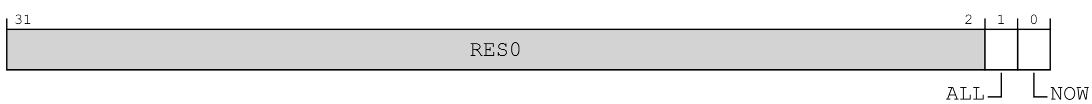
This register is present only when FEAT_MPAM is implemented, MPAMF_IDR.HAS_MSMON == 1 and MPAMF_MSMON_IDR.HAS_LOCAL_CAPT_EVNT == 1. Otherwise, direct accesses to MSMON_CAPT_EVNT are RES0.

The power and reset domain of each MSC component is specific to that component.

Attributes

MSMON_CAPT_EVNT is a 32-bit register.

Field descriptions



Bits [31:2]

Reserved, RES0.

ALL, bit [1]

In the Secure instance of this register:

- If ALL is 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 ALL is 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.

In the Root instance of this register:

- If ALL is written as 1 and NOW is also written as 1, signal a capture event to Root, Realm, Secure, and Non-secure monitor instances in this MSC that are configured with CAPT_EVNT = 7.
- If ALL is written as 0 and NOW is written as 1, signal a capture event to Root monitor instances within this MSC that are configured with CAPT_EVNT = 7.

In the Realm instance of this register:

- If ALL is written as 1 and NOW is also written as 1, signal a capture event to Realm and Non-secure monitor instances in this MSC that are configured with CAPT_EVNT = 7.
- If ALL is written as 0 and NOW is written as 1, signal a capture event to Realm monitor instances within this MSC that are configured with CAPT_EVNT = 7.

This bit always reads as zero.

`0b0` Send capture event only to monitor instances in the same MPAM feature page as this register.

`0b1` Send capture event to monitor instances in certain MPAM feature pages as described in the ALL field of this register.

NOW, bit [0]

When written as 1, this bit causes an event to those monitor instances described in the ALL field that have 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_EVNT_ns must be accessible from the Non-secure MPAM feature page.

MSMON_CAPT_EVNT_s and MSMON_CAPT_EVNT_ns must be separate registers. The Secure instance (MSMON_CAPT_EVNT_s) can generate local capture events for Secure monitor instances only or for Secure and Non-secure monitor instances, and the Non-secure instance (MSMON_CAPT_EVNT_ns) can generate local capture events for Non-secure monitor instances 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.

MSMON_CAPT_EVNT can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rt	`0x0808`	MSMON_CAPT_EVNT_rt

This interface is accessible as follows:

- When FEAT_RME is implemented accesses to this register are RW.

MSMON_CAPT_EVNT can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rl	`0x0808`	MSMON_CAPT_EVNT_rl

This interface is accessible as follows:

- When FEAT_RME is implemented 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

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. MSMON_CFG_CSU_CTL_rt controls the monitor configuration for the Root PARTID selected by the Root instance of MSMON_CFG_MON_SEL. MSMON_CFG_CSU_CTL_rl controls the monitor configuration for the Realm PARTID selected by the Realm instance of MSMON_CFG_MON_SEL.

If MPAMF_IDR.HAS_RIS is 1, the monitor instance configuration accessed is for the resource instance currently selected by MSMON_CFG_MON_SEL.RIS and the monitor instance of that resource instance selected by MSMON_CFG_MON_SEL.MON_SEL.

Configurations

The power domain of MSMON_CFG_CSU_CTL is IMPLEMENTATION DEFINED.

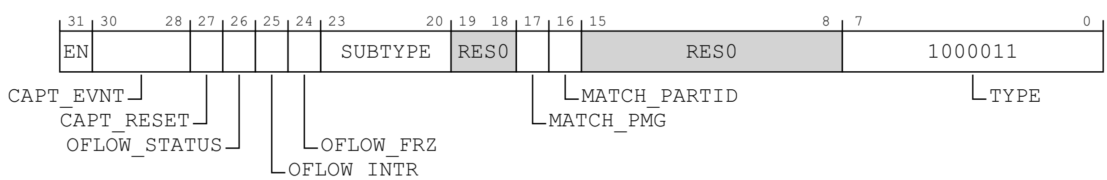
This register is present only when FEAT_MPAM is implemented, MPAMF_IDR.HAS_MSMON == 1 and MPAMF_MSMON_IDR.MSMON_CSU == 1. Otherwise, direct accesses to MSMON_CFG_CSU_CTL are RES0.

The power and reset domain of each MSC component is specific to that component.

Attributes

MSMON_CFG_CSU_CTL is a 32-bit register.

Field descriptions



EN, bit [31]

Enabled.

`0b0` The monitor instance is disabled and must not collect any information.

`0b1` The monitor instance is enabled to collect information according to the configuration of the instance.

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 must 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.

Controls whether an overflow interrupt is generated when 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. Type of cache storage usage counted by this monitor.

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, MSMON_CSU.VALUE 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. The CSU monitor is TYPE = `0x43`.

TYPE is a read-only constant indicating the type of the monitor.

Reads as `0x43`.

Access to this field is RO.

Accessing the MSMON_CFG_CSU_CTL:

This register is within the MPAM feature page memory frames.

In a system that supports Secure, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps.

- 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_rt must be accessible from the Root MPAM feature page.
- MSMON_CFG_CSU_CTL_rl must be accessible from the Realm MPAM feature page.

MSMON_CFG_CSU_CTL_s, MSMON_CFG_CSU_CTL_ns, MSMON_CFG_CSU_CTL_rt, and MSMON_CFG_CSU_CTL_rl must be separate registers.

- The Secure instance (MSMON_CFG_CSU_CTL_s) accesses the cache storage usage monitor controls used for Secure PARTIDs.
- The Non-secure instance (MSMON_CFG_CSU_CTL_ns) accesses the cache storage usage monitor controls used for Non-secure PARTIDs.
- The Root instance (MSMON_CFG_CSU_CTL_rt) accesses the cache storage usage monitor controls used for Root PARTIDs.

- The Realm instance (MSMON_CFG_CSU_CTL_rl) accesses the cache storage usage monitor controls used for Realm PARTIDs.

When RIS is implemented, loads and stores to MSMON_CFG_CSU_CTL access the cache storage usage monitor configuration settings for the cache resource instance selected by MSMON_CFG_MON_SEL.RIS and the cache storage usage monitor instance selected by MSMON_CFG_MON_SEL.MON_SEL.

When RIS is not implemented, loads and stores to MSMON_CFG_CSU_CTL access the cache storage usage monitor configuration settings for the cache storage usage monitor instance selected by MSMON_CFG_MON_SEL.MON_SEL.

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.

MSMON_CFG_CSU_CTL can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rt	`0x0818`	MSMON_CFG_CSU_CTL_rt

This interface is accessible as follows:

- When FEAT_RME is implemented 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_rl	`0x0818`	MSMON_CFG_CSU_CTL_rl

This interface is accessible as follows:

- When FEAT_RME is implemented 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

Configures 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. MSMON_CFG_CSU_FLT_rt sets the filter conditions for the Root PARTID selected by the Root instance of MSMON_CFG_MON_SEL. MSMON_CFG_CSU_FLT_rl sets the filter conditions for the Realm PARTID selected by the Realm instance of MSMON_CFG_MON_SEL.

If MPAMF_IDR.HAS_RIS is 1, the monitor instance filter configuration accessed is for the resource instance currently selected by MSMON_CFG_MON_SEL.RIS and the monitor instance of that resource instance selected by MSMON_CFG_MON_SEL.MON_SEL.

Configurations

The power domain of MSMON_CFG_CSU_FLT is IMPLEMENTATION DEFINED.

This register is present only when FEAT_MPAM is implemented, MPAMF_IDR.HAS_MSMON == 1 and MPAMF_MSMON_IDR.MSMON_CSU == 1. Otherwise, direct accesses to MSMON_CFG_CSU_FLT are RES0.

The power and reset domain of each MSC component is specific to that component.

Attributes

MSMON_CFG_CSU_FLT is a 32-bit register.

Field descriptions

31 – 24	23 – 16	15 – 0
RES0	PMG	PARTID

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 behavior of the monitor 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, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps.

- 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_rt must be accessible from the Root MPAM feature page.
- MSMON_CFG_CSU_FLT_rl must be accessible from the Realm MPAM feature page.

MSMON_CFG_CSU_FLT_s, MSMON_CFG_CSU_FLT_ns, MSMON_CFG_CSU_FLT_rt, and MSMON_CFG_CSU_FLT_rl 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.
- 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.
- The Root instance (MSMON_CFG_CSU_FLT_rt) accesses the PARTID and PMG matching for a cache storage usage monitor used for Root PARTIDs.
- The Realm instance (MSMON_CFG_CSU_FLT_rl) accesses the PARTID and PMG matching for a cache storage usage monitor used for Realm PARTIDs.

When RIS is implemented, loads and stores to MSMON_CFG_CSU_FLT access the monitor configuration settings for the resource instance selected by MSMON_CFG_MON_SEL.RIS and the cache storage usage monitor instance selected by MSMON_CFG_MON_SEL.MON_SEL.

When RIS is not implemented, loads and stores to MSMON_CFG_CSU_FLT access the monitor configuration settings for the cache storage usage monitor instance selected by MSMON_CFG_MON_SEL.MON_SEL.

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.

MSMON_CFG_CSU_FLT can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rt	`0x0810`	MSMON_CFG_CSU_FLT_rt

This interface is accessible as follows:

- When FEAT_RME is implemented 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_rl	`0x0810`	MSMON_CFG_CSU_FLT_rl

This interface is accessible as follows:

- When FEAT_RME is implemented 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

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. MSMON_CFG_MBWU_CTL_rt controls the monitor configuration for the Root PARTID selected by the Root instance of MSMON_CFG_MON_SEL. MSMON_CFG_MBWU_CTL_rl controls the monitor configuration for the Realm PARTID selected by the Realm instance of MSMON_CFG_MON_SEL.

If MPAMF_IDR.HAS_RIS is 1, the monitor instance configuration accessed is for the resource instance currently selected by MSMON_CFG_MON_SEL.RIS and the monitor instance of that resource instance selected by MSMON_CFG_MON_SEL.MON_SEL.

Configurations

The power domain of MSMON_CFG_MBWU_CTL is IMPLEMENTATION DEFINED.

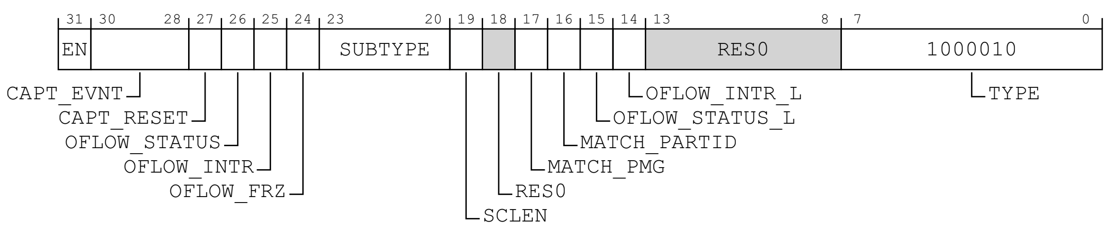
This register is present only when FEAT_MPAM is implemented, MPAMF_IDR.HAS_MSMON == 1 and MPAMF_MSMON_IDR.MSMON_MBWU == 1. Otherwise, direct accesses to MSMON_CFG_MBWU_CTL are RES0.

The power and reset domain of each MSC component is specific to that component.

Attributes

MSMON_CFG_MBWU_CTL is a 32-bit register.

Field descriptions



EN, bit [31]

Enabled.

`0b0` The monitor instance is disabled and must not collect any information.

`0b1` The monitor instance is enabled to collect information according to the configuration of the instance.

CAPT_EVNT, bits [30:28]

Capture event selector.

When the selected capture event occurs, MSMON_MBWU of the monitor instance is copied to MSMON_MBWU_CAPTURE of the same instance. If the long counter is also implemented, MSMON_MBWU_L is also copied to MSMON_MBWU_L_CAPTURE.

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 must 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 MSMON_MBWU.VALUE after capture.

Controls whether the VALUE field of the monitor instance is reset to zero immediately after being copied to the corresponding capture register.

`0b0` MSMON_MBWU.VALUE field of the monitor instance is not reset on capture.

`0b1` MSMON_MBWU.VALUE field of the monitor instance 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.

This control bit affects both MSMON_MBWU and MSMON_MBWU_L in implementations that include MSMON_MBWU_L.

OFLOW_STATUS, bit [26]

Overflow status.

Indicates whether the value of MSMON_MBWU has overflowed.

`0b0` MSMON_MBWU.VALUE has not overflowed.

`0b1` MSMON_MBWU.VALUE has overflowed at least once since this bit was last written to zero.

If overflow is not possible for an MBWU monitor in the MSC implementation, this field is RAZ/WI.

Overflow status for MSMON_MBWU_L.VALUE is reported in MSMON_CFG_MBWU_CTL.OFLOW_STATUS_L.

OFLOW_INTR, bit [25]

Enable interrupt on overflow of MSMON_MBWU.VALUE.

`0b0` No interrupt is signaled on an overflow of MSMON_MBWU.VALUE.

`0b1` An implementation-specific interrupt is signaled on an overflow of MSMON_MBWU.VALUE.

If overflow is not possible for an MBWU monitor in the MSC implementation, this field is RAZ/WI.

If overflow interrupt is not supported by the MSC implementation, this field is RAZ/WI.

Interrupt enable for overflow of MSMON_MBWU_L.VALUE is controlled by MSMON_CFG_MBWU_CTL.OFLOW_INTR_L.

OFLOW_FRZ, bit [24]

Freeze monitor instance on overflow.

Controls whether MSMON_MBWU.VALUE field of the monitor instance freezes on an overflow.

`0b0` MSMON_MBWU.VALUE field of the monitor instance wraps on overflow.

`0b1` MSMON_MBWU.VALUE field of the monitor instance freezes on overflow. If the increment that caused the overflow was 1, the frozen value is the post-increment value of 0. If the increment that caused the overflow was larger than 1, the frozen value of the monitor might be 0 or a larger value less than the final increment.

If overflow is not possible for the instance of the MBWU monitor in the implementation, this field is RAZ/WI.

This control bit affects both MSMON_MBWU and MSMON_MBWU_L in implementations that include MSMON_MBWU_L.

SUBTYPE, bits [23:20]

Subtype. Type of bandwidth counted by this monitor.

This field is not currently used for MBWU monitors, but reserved for future use.

This field is RAZ/WI.

SCLEN, bit [19]

MSMON_MBWU.VALUE Scaling Enable.

Enables scaling of MSMON_MBWU.VALUE by MPAMF_MBWUMON_IDR.SCALE.

`0b0` MSMON_MBWU.VALUE has bytes counted by the monitor instance.

`0b1` MSMON_MBWU.VALUE has bytes counted by the monitor instance, shifted right by MPAMF_MBWUMON_IDR.SCALE.

Bit [18]

Reserved, RES0.

MATCH_PMG, bit [17]

Match PMG.

Controls whether the monitor instance only counts data transferred with PMG matching MSMON_CFG_MBWU_FLT.PMG.

`0b0` The monitor instance counts data transferred with any PMG value.

`0b1` The monitor instance only counts data transferred with the PMG value matching MSMON_CFG_MBWU_FLT.PMG.

MATCH_PARTID, bit [16]

Match PARTID.

Controls whether the monitor instance counts only data transferred with PARTID matching MSMON_CFG_MBWU_FLT.PARTID.

`0b0` The monitor instance counts data transferred with any PARTID value.

`0b1` The monitor instance only counts data transferred with the PARTID value matching MSMON_CFG_MBWU_FLT.PARTID.

OFLOW_STATUS_L, bit [15]

When FEAT_MPAMv0p1 is implemented or FEAT_MPAMv1p1 is implemented:

Overflow Status of MSMON_MBWU_L.VALUE of the monitor instance.

Indicates whether MSMON_MBWU_L.VALUE has overflowed.

`0b0` MSMON_MBWU_L.VALUE has not overflowed.

`0b1` MSMON_MBWU_L.VALUE has overflowed at least once since this bit was last written to zero.

If MPAMF_MBWUMON_IDR.HAS_LONG == 0, this bit is RES0.

Overflow status of MSMON_MBWU.VALUE is reported in MSMON_CFG_MBWU_CTL.OFLOW_STATUS.

Otherwise:

Reserved, RES0.

OFLOW_INTR_L, bit [14]

When FEAT_MPAMv0p1 is implemented or FEAT_MPAMv1p1 is implemented:

Overflow Interrupt for MSMON_MBWU_L.

Controls whether an MPAM overflow interrupt is generated when MSMON_MBWU_L.VALUE overflows.

`0b0`	No interrupt is signaled on an overflow of MSMON_MBWU_L.VALUE.
`0b1`	An implementation-specific interrupt is signaled on overflow of MSMON_MBWU_L.VALUE.

If overflow is not possible for an MBWU monitor in the MSC implementation, this field is RAZ/WI.

If the overflow interrupt is not supported by the MSC implementation, this field is RAZ/WI.

If MPAMF_MBWUMON_IDR.HAS_LONG == 0, this bit is RES0.

Otherwise:

Reserved, RES0.

Bits [13:8]

Reserved, RES0.

TYPE, bits [7:0]

Monitor Type Code. The MBWU monitor is TYPE = `0x42`.

TYPE is a read-only constant indicating the type of the monitor.

Reads as `0x42`.

Access to this field is RO.

Accessing the MSMON_CFG_MBWU_CTL:

This register is within the MPAM feature page memory frames.

In a system that supports Secure, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps.

- 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_rt must be accessible from the Root MPAM feature page.
- MSMON_CFG_MBWU_CTL_rl must be accessible from the Realm MPAM feature page.

MSMON_CFG_MBWU_CTL_s, MSMON_CFG_MBWU_CTL_ns, MSMON_CFG_MBWU_CTL_rt, and MSMON_CFG_MBWU_CTL_rl must be separate registers.

- The Secure instance (MSMON_CFG_MBWU_CTL_s) accesses the memory bandwidth usage monitor controls used for Secure PARTIDs.
- The Non-secure instance (MSMON_CFG_MBWU_CTL_ns) accesses the memory bandwidth usage monitor controls used for Non-secure PARTIDs.
- The Root instance (MSMON_CFG_MBWU_CTL_rt) accesses the memory bandwidth usage monitor controls used for Root PARTIDs.
- The Realm instance (MSMON_CFG_MBWU_CTL_rl) accesses the memory bandwidth usage monitor controls used for Realm PARTIDs.

When RIS is implemented, loads and stores to MSMON_CFG_MBWU_CTL access the monitor configuration settings for the bandwidth resource instance selected by MSMON_CFG_MON_SEL.RIS and the memory bandwidth usage monitor instance selected by MSMON_CFG_MON_SEL.MON_SEL.

When RIS is not implemented, loads and stores to MSMON_CFG_MBWU_CTL access the monitor configuration settings for the memory bandwidth usage monitor instance selected by MSMON_CFG_MON_SEL.MON_SEL.

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.

MSMON_CFG_MBWU_CTL can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rt	0x0828	MSMON_CFG_MBWU_CTL_rt

This interface is accessible as follows:

- When FEAT_RME is implemented 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_rl	0x0828	MSMON_CFG_MBWU_CTL_rl

This interface is accessible as follows:

- When FEAT_RME is implemented 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

Controls 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. MSMON_CFG_CSU_FLT_rt sets the filter conditions for the Root PARTID selected by the Root instance of MSMON_CFG_MON_SEL. MSMON_CFG_CSU_FLT_rl sets the filter conditions for the Realm PARTID selected by the Realm instance of MSMON_CFG_MON_SEL.

If MPAMF_IDR.HAS_RIS is 1, the monitor instance filter configuration accessed is for the resource instance currently selected by MSMON_CFG_MON_SEL.RIS and the monitor instance of that resource instance selected by MSMON_CFG_MON_SEL.MON_SEL.

Configurations

The power domain of MSMON_CFG_MBWU_FLT is IMPLEMENTATION DEFINED.

This register is present only when FEAT_MPAM is implemented, MPAMF_IDR.HAS_MSMON == 1 and MPAMF_MSMON_IDR.MSMON_MBWU == 1. Otherwise, direct accesses to MSMON_CFG_MBWU_FLT are RES0.

The power and reset domain of each MSC component is specific to that component.

Attributes

MSMON_CFG_MBWU_FLT is a 32-bit register.

Field descriptions

When FEAT_MPAMv0p1 is implemented or FEAT_MPAMv1p1 is implemented:

31 30	29 24	23 16	15 0
RWBW	RES0	PMG	PARTID

RW filtering.

RWBW, bits [31:30]

When MPAMF_MBWUMON_IDR.HAS_RWBW == 1:

Read/write bandwidth filter. Configures the selected monitor instance to count all bandwidth, only read bandwidth or only write bandwidth.

`0b00`	Monitor instance counts read bandwidth and write bandwidth.
`0b01`	Monitor instance counts write bandwidth only.
`0b10`	Monitor instance counts read bandwidth only.
`0b11`	Reserved.

Otherwise:

Reserved, RES0.

Bits [29:24]

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.

Otherwise:

31 – 24	23 – 16	15 – 0
RES0	PMG	PARTID

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, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps.

- 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_rt must be accessible from the Root MPAM feature page.
- MSMON_CFG_MBWU_FLT_rl must be accessible from the Realm MPAM feature page.

MSMON_CFG_MBWU_FLT_s, MSMON_CFG_MBWU_FLT_ns, MSMON_CFG_MBWU_FLT_rt, and MSMON_CFG_MBWU_FLT_rl 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.
- 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.
- The Root instance (MSMON_CFG_MBWU_FLT_rt) accesses the PARTID and PMG matching for a memory bandwidth usage monitor used for Root PARTIDs.
- The Realm instance (MSMON_CFG_MBWU_FLT_rl) accesses the PARTID and PMG matching for a memory bandwidth usage monitor used for Realm PARTIDs.

When RIS is implemented, loads and stores to MSMON_CFG_MBWU_FLT access the monitor configuration settings for the bandwidth resource instance selected by MSMON_CFG_MON_SEL.RIS and the memory bandwidth usage monitor instance selected by MSMON_CFG_MON_SEL.MON_SEL.

When RIS is not implemented, loads and stores to MSMON_CFG_MBWU_FLT access the monitor configuration settings for the memory bandwidth usage monitor instance selected by MSMON_CFG_MON_SEL.MON_SEL.

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.

MSMON_CFG_MBWU_FLT can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rt	0x0820	MSMON_CFG_MBWU_FLT_rt

This interface is accessible as follows:

- When FEAT_RME is implemented 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_rl	0x0820	MSMON_CFG_MBWU_FLT_rl

This interface is accessible as follows:

- When FEAT_RME is implemented 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. MSMON_CFG_MON_SEL_rt selects a Root monitor instance to access via the Root MPAM feature page. MSMON_CFG_MON_SEL_rl selects a Realm monitor instance to access via the Realm MPAM feature page.

Note

Different performance monitoring features within an MSC could have different numbers of monitor instances. See the NUM_MON field in the corresponding ID register. This means that 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 FEAT_MPAM is implemented and (MPAMF_IDR.HAS_MSMON == 1, or (MPAMF_IDR.HAS_IMPL_IDR == 1 and MPAMF_IDR.EXT == 0) or (MPAMF_IDR.HAS_IMPL_IDR == 1, MPAMF_IDR.EXT == 1 and MPAMF_IDR.NO_IMPL_MSMON == 0)). Otherwise, direct accesses to MSMON_CFG_MON_SEL are RES0.

The power and reset domain of each MSC component is specific to that component.

Attributes

MSMON_CFG_MON_SEL is a 32-bit register.

Field descriptions

31:28	27:24	23:16	15:0
RES0	RIS	RES0	MON_SEL

Bits [31:28]

Reserved, RES0.

RIS, bits [27:24]

When (FEAT_MPAMv0p1 is implemented or FEAT_MPAMv1p1 is implemented), MPAMF_IDR.EXT == 1 and MPAMF_IDR.HAS_RIS == 1:

Resource Instance Selector. RIS selects one resource to configure through MSMON_CFG registers.

Otherwise:

Reserved, RES0.

Bits [23:16]

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 within the MPAM feature page memory frames.

In a system that supports Secure, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps.

- MSMON_CFG_MON_SEL_s must be accessible from the Secure MPAM feature page.
- MSMON_CFG_MON_SEL_ns must be accessible from the Non-secure MPAM feature page.
- MSMON_CFG_MON_SEL_rt must be accessible from the Root MPAM feature page.
- MSMON_CFG_MON_SEL_rl must be accessible from the Realm MPAM feature page.

MSMON_CFG_MON_SEL_s, MSMON_CFG_MON_SEL_ns, MSMON_CFG_MON_SEL_rt, and MSMON_CFG_MON_SEL_rl must be separate registers.

- The Secure instance (MSMON_CFG_MON_SEL_s) accesses the monitor instance selector used for Secure PARTIDs.
- The Non-secure instance (MSMON_CFG_MON_SEL_ns) accesses the monitor instance selector used for Non-secure PARTIDs.
- The Root instance (MSMON_CFG_MON_SEL_rt) accesses the monitor instance selector used for Root PARTIDs.
- The Realm instance (MSMON_CFG_MON_SEL_rl) accesses the monitor instance selector used for Realm 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.

MSMON_CFG_MON_SEL can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rt	`0x0800`	MSMON_CFG_MON_SEL_rt

This interface is accessible as follows:

- When FEAT_RME is implemented 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_rl	0x0800	MSMON_CFG_MON_SEL_rl

This interface is accessible as follows:

- When FEAT_RME is implemented accesses to this register are RW.

11.5.7 MSMON_CSU, MPAM Cache Storage Usage Monitor Register

The MSMON_CSU characteristics are:

Purpose

Accesses the CSU monitor instance selected by MSMON_CFG_MON_SEL.

MSMON_CSU_s is a Secure cache storage usage monitor instance selected by the Secure instance of MSMON_CFG_MON_SEL. MSMON_CSU_ns is a Non-secure cache storage usage monitor instance selected by the Non-secure instance of MSMON_CFG_MON_SEL. MSMON_CSU_rt is a Root cache storage usage monitor instance selected by the Root instance of MSMON_CFG_MON_SEL. MSMON_CSU_rl is a Realm cache storage usage monitor instance selected by the Realm instance of MSMON_CFG_MON_SEL.

If MPAMF_IDR.HAS_RIS is 1, the monitor instance accessed is for the resource instance currently selected by MSMON_CFG_MON_SEL.RIS and the monitor instance of that resource instance selected by MSMON_CFG_MON_SEL.MON_SEL.

Configurations

The power domain of MSMON_CSU is IMPLEMENTATION DEFINED.

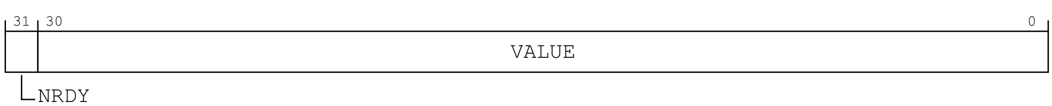
This register is present only when FEAT_MPAM is implemented, MPAMF_IDR.HAS_MSMON == 1 and MPAMF_MSMON_IDR.MSMON_CSU == 1. Otherwise, direct accesses to MSMON_CSU are RES0.

The power and reset domain of each MSC component is specific to that component.

Attributes

MSMON_CSU is a 32-bit register.

Field descriptions



NRDY, bit [31]

Not Ready. Indicates whether the monitor instance has possibly inaccurate data.

`0b0` The monitor instance is ready and the MSMON_CSU.VALUE field is accurate.

`0b1` The monitor instance is not ready and the contents of the MSMON_CSU.VALUE field might be inaccurate or otherwise not represent the actual cache storage usage.

VALUE, bits [30:0]

Cache storage usage measurement value if MSMON_CSU.NRDY is 0. Invalid if MSMON_CSU.NRDY is 1.

VALUE is the cache storage usage measured 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, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps.

- 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_rt must be accessible from the Root MPAM feature page.
- MSMON_CSU_rl must be accessible from the Realm MPAM feature page.

MSMON_CSU_s, MSMON_CSU_ns, MSMON_CSU_rt, and MSMON_CSU_rl must be separate registers.

- The Secure instance (MSMON_CSU_s) accesses the cache storage usage monitor used for Secure PARTIDs.
- The Non-secure instance (MSMON_CSU_ns) accesses the cache storage usage monitor used for Non-secure PARTIDs.
- The Root instance (MSMON_CSU_rt) accesses the cache storage usage monitor used for Root PARTIDs.
- The Realm instance (MSMON_CSU_rl) accesses the cache storage usage monitor used for Realm PARTIDs.

When RIS is implemented, reads and writes to MSMON_CSU access the cache storage usage monitor monitor instance for the cache resource instance selected by MSMON_CFG_MON_SEL.RIS and the cache storage usage monitor instance selected by MSMON_CFG_MON_SEL.MON_SEL.

When RIS is not implemented, reads and writes to MSMON_CSU access the cache storage usage monitor monitor instance for the cache storage usage monitor instance selected by MSMON_CFG_MON_SEL.MON_SEL.

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:

- When MPAMF_CSUMON_IDR.CSU_RO == 0 accesses to this register are RW.
- When MPAMF_CSUMON_IDR.CSU_RO == 1 accesses to this register are RO.

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:

- When MPAMF_CSUMON_IDR.CSU_RO == 0 accesses to this register are RW.
- When MPAMF_CSUMON_IDR.CSU_RO == 1 accesses to this register are RO.

MSMON_CSU can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rt	`0x0840`	MSMON_CSU_rt

This interface is accessible as follows:

- When FEAT_RME is implemented and MPAMF_CSUMON_IDR.CSU_RO == 0 accesses to this register are RW.
- When FEAT_RME is implemented and MPAMF_CSUMON_IDR.CSU_RO == 1 accesses to this register are RO.

MSMON_CSU can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rl	0x0840	MSMON_CSU_rl

This interface is accessible as follows:

- When FEAT_RME is implemented and MPAMF_CSUMON_IDR.CSU_RO == 0 accesses to this register are RW.
- When FEAT_RME is implemented and MPAMF_CSUMON_IDR.CSU_RO == 1 accesses to this register are RO.

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 instance 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. MSMON_CSU_CAPTURE_rt is a Root cache storage usage monitor capture instance selected by the Root instance of MSMON_CFG_MON_SEL. MSMON_CSU_CAPTURE_rl is a Realm cache storage usage monitor capture instance selected by the Realm instance of MSMON_CFG_MON_SEL.

If MPAMF_IDR.HAS_RIS is 1, the monitor instance capture register accessed is for the resource instance currently selected by MSMON_CFG_MON_SEL.RIS and the monitor instance of that resource instance selected by MSMON_CFG_MON_SEL.MON_SEL.

Configurations

The power domain of MSMON_CSU_CAPTURE is IMPLEMENTATION DEFINED.

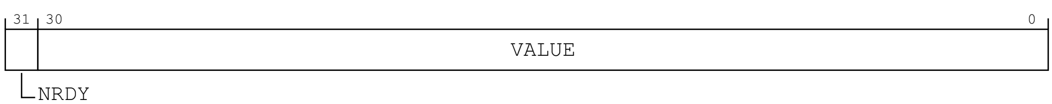
This register is present only when FEAT_MPAM is implemented, MPAMF_IDR.HAS_MSMON == 1, MPAMF_MSMON_IDR.MSMON_CSU == 1 and MPAMF_CSUMON_IDR.HAS_CAPTURE == 1. Otherwise, direct accesses to MSMON_CSU_CAPTURE are RES0.

The power and reset domain of each MSC component is specific to that component.

Attributes

MSMON_CSU_CAPTURE is a 32-bit register.

Field descriptions



NRDY, bit [31]

Not Ready. Indicates whether the captured monitor value has possibly inaccurate data.

`0b0` The captured monitor instance was ready and the MSMON_CSU_CAPTURE.VALUE field is accurate.

`0b1` The captured monitor instance was not ready and the contents of the MSMON_CSU_CAPTURE.VALUE field might be inaccurate or otherwise not represent the actual cache storage usage.

VALUE, bits [30:0]

Captured cache storage usage measurement if MSMON_CSU_CAPTURE.NRDY is 0. Invalid if MSMON_CSU_CAPTURE.NRDY is 1.

VALUE is the captured cache storage usage measurement 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, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps.

- 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_rt must be accessible from the Root MPAM feature page.
- MSMON_CSU_CAPTURE_rl must be accessible from the Realm MPAM feature page.

MSMON_CSU_CAPTURE_s, MSMON_CSU_CAPTURE_ns, MSMON_CSU_CAPTURE_rt, and MSMON_CSU_CAPTURE_rl must be separate registers.

- The Secure instance (MSMON_CSU_CAPTURE_s) accesses the captured cache storage usage monitor used for Secure PARTIDs.
- The Non-secure instance (MSMON_CSU_CAPTURE_ns) accesses the captured cache storage usage monitor used for Non-secure PARTIDs.
- The Root instance (MSMON_CSU_CAPTURE_rt) accesses the captured cache storage usage monitor used for Root PARTIDs.
- The Realm instance (MSMON_CSU_CAPTURE_rl) accesses the captured cache storage usage monitor used for Realm PARTIDs.

When RIS is implemented, reads and writes to MSMON_CSU_CAPTURE access the monitor instance for the cache resource instance selected by MSMON_CFG_MON_SEL.RIS and the cache storage usage monitor instance selected by MSMON_CFG_MON_SEL.MON_SEL.

When RIS is not implemented, reads and writes to MSMON_CSU_CAPTURE access the monitor instance for the cache storage usage monitor instance selected by MSMON_CFG_MON_SEL.MON_SEL.

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.

MSMON_CSU_CAPTURE can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rt	0x0848	MSMON_CSU_CAPTURE_rt

This interface is accessible as follows:

- When FEAT_RME is implemented accesses to this register are RW.

MSMON_CSU_CAPTURE can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rl	`0x0848`	MSMON_CSU_CAPTURE_rl

This interface is accessible as follows:

- When FEAT_RME is implemented accesses to this register are RW.

11.5.9 MSMON_CSU_OFSR, MPAM CSU Monitor Overflow Status Register

The MSMON_CSU_OFSR characteristics are:

Purpose

MSMON_CSU_OFSR is a 32-bit read-only register that shows bitmap of CSU monitor instance overflow status for a contiguous group of 32 monitor instances.

MSMON_CSU_OFSR_s gives a bitmap of pending CSU overflow status for 32 Secure CSU monitor instances. MSMON_CSU_OFSR_ns gives a bitmap of pending CSU overflow status for 32 Non-secure CSU monitor instances. MSMON_CSU_OFSR_rt gives a bitmap of pending CSU overflow status for 32 Root CSU monitor instances. MSMON_CSU_OFSR_rl gives a bitmap of pending CSU overflow status for 32 Realm CSU monitor instances.

Configurations

The power domain of MSMON_CSU_OFSR is IMPLEMENTATION DEFINED.

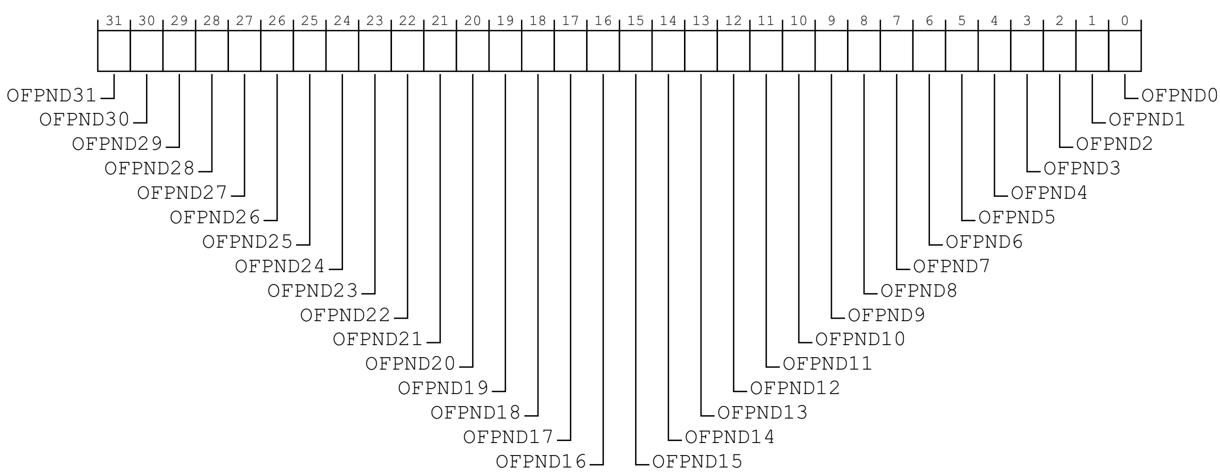
This register is present only when MPAMF_CSUMON_IDR.HAS_OFSR == 1. Otherwise, direct accesses to MSMON_CSU_OFSR are RES0.

The power and reset domain of each MSC component is specific to that component.

Attributes

MSMON_CSU_OFSR is a 32-bit register.

Field descriptions



OFPND<i>, bit [i], for i = 31 to 0

Overflow status bitmap for CSU monitor instances. The RIS and the contiguous range of CSU monitor instances are set in MSMON_CFG_MON_SEL. i of 0 corresponds to the CSU monitor instance MSMON_CFG_MON_SEL.MON_SEL & `0xFFE0`.

`0b0` CSU monitor instance (MSMON_CFG_MON_SEL.MON_SEL & `0xFFE0` + i) does not have a pending overflow.

`0b1` CSU monitor instance (MSMON_CFG_MON_SEL.MON_SEL & `0xFFE0` + i) has a pending overflow.

After reading MSMON_OFLOW_SR to determine that a CSU monitor instance has a pending overflow and which RIS values have pending overflows, an interrupt service routine could poll groups of 32 monitor instances in a RIS for pending monitors by reading this bitmap and incrementing MSMON_CFG_MON_SEL.MON_SEL by 32.

Accessing the MSMON_CSU_OFSR:

This register is within the MPAM feature page memory frames.

In a system that supports Secure, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps.

- MSMON_CSU_OFSR_s must be accessible from the Secure MPAM feature page.
- MSMON_CSU_OFSR_ns must be accessible from the Non-secure MPAM feature page.
- MSMON_CSU_OFSR_rt must be accessible from the Root MPAM feature page.
- MSMON_CSU_OFSR_rl must be accessible from the Realm MPAM feature page.

MSMON_CSU_OFSR_s, MSMON_CSU_OFSR_ns, MSMON_CSU_OFSR_rt, and MSMON_CSU_OFSR_rl must be separate registers.

- The Secure instance (MSMON_CSU_OFSR_s) accesses the CSU monitor overflow status bitmap used for Secure PARTIDs.
- The Non-secure instance (MSMON_CSU_OFSR_ns) accesses the CSU monitor overflow status bitmap used for Non-secure PARTIDs.
- The Root instance (MSMON_CSU_OFSR_rt) accesses the CSU monitor overflow status bitmap used for Root PARTIDs.
- The Realm instance (MSMON_CSU_OFSR_rl) accesses the CSU monitor overflow status bitmap used for Realm PARTIDs.

MSMON_CSU_OFSR can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_s	`0x0858`	MSMON_CSU_OFSR_s

This interface is accessible as follows:

- Accesses to this register are RO.

MSMON_CSU_OFSR can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_ns	`0x0858`	MSMON_CSU_OFSR_ns

This interface is accessible as follows:

- Accesses to this register are RO.

MSMON_CSU_OFSR can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rt	`0x0858`	MSMON_CSU_OFSR_rt

This interface is accessible as follows:

- When FEAT_RME is implemented accesses to this register are RO.

MSMON_CSU_OFSR can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rl	`0x0858`	MSMON_CSU_OFSR_rl

This interface is accessible as follows:

- When FEAT_RME is implemented accesses to this register are RO.

11.5.10 MSMON_MBWU, MPAM Memory Bandwidth Usage Monitor Register

The MSMON_MBWU characteristics are:

Purpose

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. MSMON_MBWU_rt is the Root memory bandwidth usage monitor instance selected by MSMON_CFG_MON_SEL_rt. MSMON_MBWU_rl is the Realm memory bandwidth usage monitor instance selected by MSMON_CFG_MON_SEL_rl.

If MPAMF_IDR.HAS_RIS is 1, the monitor instance register accessed is for the resource instance currently selected by MSMON_CFG_MON_SEL.RIS and the monitor instance of that resource instance selected by MSMON_CFG_MON_SEL.MON_SEL.

Configurations

The power domain of MSMON_MBWU is IMPLEMENTATION DEFINED.

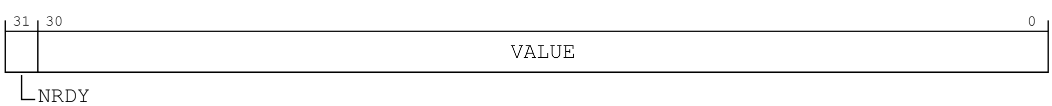
This register is present only when FEAT_MPAM is implemented, MPAMF_IDR.HAS_MSMON == 1 and MPAMF_MSMON_IDR.MSMON_MBWU == 1. Otherwise, direct accesses to MSMON_MBWU are RES0.

The power and reset domain of each MSC component is specific to that component.

Attributes

MSMON_MBWU is a 32-bit register.

Field descriptions



NRDY, bit [31]

Not Ready. Indicates whether the monitor has possibly inaccurate data.

`0b0` The monitor instance is ready and the MSMON_MBWU.VALUE field is accurate.

`0b1` The monitor instance is not ready and the contents of the MSMON_MBWU.VALUE field might be inaccurate or otherwise not represent the actual memory bandwidth usage.

VALUE, bits [30:0]

Memory bandwidth usage counter value if MSMON_MBWU.NRDY is 0. Invalid if MSMON_MBWU.NRDY is 1.

VALUE is the scaled count of bytes transferred since the monitor was last reset that met 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 enables scaling, the count in VALUE is the number of bytes shifted right by MPAMF_MBWUMON_IDR.SCALE bit positions and rounded.

Accessing the MSMON_MBWU:

This register is within the MPAM feature page memory frames.

In a system that supports Secure, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps.

- 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_rt must be accessible from the Root MPAM feature page.
- MSMON_MBWU_rl must be accessible from the Realm MPAM feature page.

MSMON_MBWU_s, MSMON_MBWU_ns, MSMON_MBWU_rt, and MSMON_MBWU_rl must be separate registers.

- The Secure instance (MSMON_MBWU_s) accesses the memory bandwidth usage monitor used for Secure PARTIDs.
- The Non-secure instance (MSMON_MBWU_ns) accesses the memory bandwidth usage monitor used for Non-secure PARTIDs.
- The Root instance (MSMON_MBWU_rt) accesses the memory bandwidth usage monitor used for Root PARTIDs.
- The Realm instance (MSMON_MBWU_rl) accesses the memory bandwidth usage monitor used for Realm PARTIDs.

When RIS is implemented, reads and writes to MSMON_MBWU access the memory bandwidth usage monitor instance for the resource instance selected by MSMON_CFG_MON_SEL.RIS and the memory bandwidth usage monitor instance selected by MSMON_CFG_MON_SEL.MON_SEL.

When RIS is not implemented, reads and writes to MSMON_MBWU access the memory bandwidth usage monitor instance for the memory bandwidth usage monitor instance selected by MSMON_CFG_MON_SEL.MON_SEL.

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.

MSMON_MBWU can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rt	`0x0860`	MSMON_MBWU_rt

This interface is accessible as follows:

- When FEAT_RME is implemented accesses to this register are RW.

MSMON_MBWU can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rl	`0x0860`	MSMON_MBWU_rl

This interface is accessible as follows:

- When FEAT_RME is implemented accesses to this register are RW.

11.5.11 MSMON_MBWU_CAPTURE, MPAM Memory Bandwidth Usage Monitor Capture Register

The MSMON_MBWU_CAPTURE characteristics are:

Purpose

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. MSMON_MBWU_CAPTURE_rt is the Root memory bandwidth usage monitor capture instance selected by the Root instance of MSMON_CFG_MON_SEL. MSMON_MBWU_CAPTURE_rl is the Realm memory bandwidth usage monitor capture instance selected by the Realm instance of MSMON_CFG_MON_SEL.

If MPAMF_IDR.HAS_RIS is 1, the monitor instance capture register accessed is for the resource instance currently selected by MSMON_CFG_MON_SEL.RIS and the monitor instance of that resource instance selected by MSMON_CFG_MON_SEL.MON_SEL.

Configurations

The power domain of MSMON_MBWU_CAPTURE is IMPLEMENTATION DEFINED.

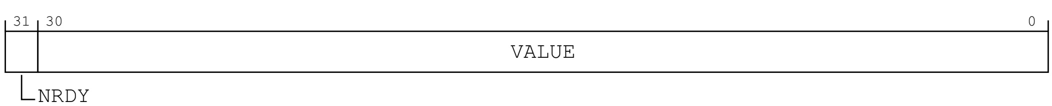
This register is present only when FEAT_MPAM is implemented, MPAMF_IDR.HAS_MSMON == 1, MPAMF_MSMON_IDR.MSMON_MBWU == 1 and MPAMF_MBWUMON_IDR.HAS_CAPTURE == 1. Otherwise, direct accesses to MSMON_MBWU_CAPTURE are RES0.

The power and reset domain of each MSC component is specific to that component.

Attributes

MSMON_MBWU_CAPTURE is a 32-bit register.

Field descriptions



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 instance was ready and the MSMON_MBWU_CAPTURE.VALUE field is accurate.

`0b1` The captured monitor instance was not ready and the contents of the MSMON_MBWU_CAPTURE.VALUE field might be inaccurate or otherwise not represent the actual memory bandwidth usage.

VALUE, bits [30:0]

Captured memory bandwidth usage counter value if MSMON_MBWU_CAPTURE.NRDY is 0. Invalid if MSMON_MBWU_CAPTURE.NRDY is 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, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps.

- 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_rt must be accessible from the Root MPAM feature page.
- MSMON_MBWU_CAPTURE_rl must be accessible from the Realm MPAM feature page.

MSMON_MBWU_CAPTURE_s, MSMON_MBWU_CAPTURE_ns, MSMON_MBWU_CAPTURE_rt, and MSMON_MBWU_CAPTURE_rl must be separate registers.

- The Secure instance (MSMON_MBWU_CAPTURE_s) accesses the captured memory bandwidth usage monitor used for Secure PARTIDs.
- The Non-secure instance (MSMON_MBWU_CAPTURE_ns) accesses the captured memory bandwidth usage monitor used for Non-secure PARTIDs.
- The Root instance (MSMON_MBWU_CAPTURE_rt) accesses the captured memory bandwidth usage monitor used for Root PARTIDs.
- The Realm instance (MSMON_MBWU_CAPTURE_rl) accesses the captured memory bandwidth usage monitor used for Realm PARTIDs.

When RIS is implemented, reads and writes to MSMON_MBWU_CAPTURE access the monitor instance for the bandwidth resource instance selected by MSMON_CFG_MON_SEL.RIS and the memory bandwidth usage monitor instance selected by MSMON_CFG_MON_SEL.MON_SEL.

When RIS is not implemented, reads and writes to MSMON_MBWU_CAPTURE access the monitor instance for the memory bandwidth usage monitor instance selected by MSMON_CFG_MON_SEL.MON_SEL.

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.

MSMON_MBWU_CAPTURE can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rt	0x0868	MSMON_MBWU_CAPTURE_rt

This interface is accessible as follows:

- When FEAT_RME is implemented accesses to this register are RW.

MSMON_MBWU_CAPTURE can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rl	`0x0868`	MSMON_MBWU_CAPTURE_rl

This interface is accessible as follows:

- When FEAT_RME is implemented accesses to this register are RW.

11.5.12 MSMON_MBWU_L, MPAM Long Memory Bandwidth Usage Monitor Register

The MSMON_MBWU_L characteristics are:

Purpose

Accesses the monitor instance selected by MSMON_CFG_MON_SEL.

MSMON_MBWU_L_s is the Secure long memory bandwidth usage monitor instance selected by the Secure instance of MSMON_CFG_MON_SEL. MSMON_MBWU_L_ns is the Non-secure long memory bandwidth usage monitor instance selected by the Non-secure instance of MSMON_CFG_MON_SEL. MSMON_MBWU_L_rt is the Root long memory bandwidth usage monitor instance selected by MSMON_CFG_MON_SEL_rt. MSMON_MBWU_L_rl is the Realm long memory bandwidth usage monitor instance selected by MSMON_CFG_MON_SEL_rl.

If MPAMF_IDR.HAS_RIS is 1, the monitor instance long monitor register accessed is for the resource instance currently selected by MSMON_CFG_MON_SEL.RIS and the monitor instance of that resource instance selected by MSMON_CFG_MON_SEL.MON_SEL.

Configurations

The power domain of MSMON_MBWU_L is IMPLEMENTATION DEFINED.

This register is present only when FEAT_MPAM is implemented, MPAMF_IDR.HAS_MSMON == 1, MPAMF_MSMON_IDR.MSMON_MBWU == 1 and MPAMF_MBWUMON_IDR.HAS_LONG == 1. Otherwise, direct accesses to MSMON_MBWU_L are RES0.

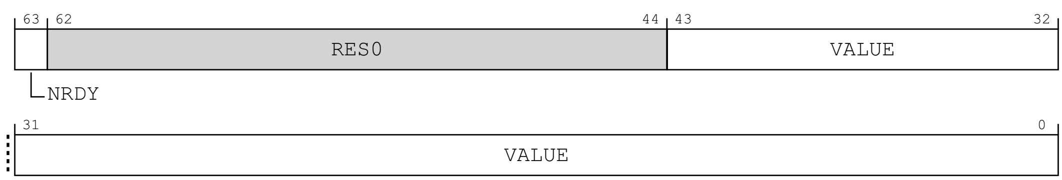
The power and reset domain of each MSC component is specific to that component.

Attributes

MSMON_MBWU_L is a 64-bit register.

Field descriptions

When MPAMF_MBWUMON_IDR.LWD == 0:



NRDY, bit [63]

Not Ready. Indicates whether the monitor instance has possibly inaccurate data.

`0b0` The monitor instance is ready and the MSMON_MBWU_L.VALUE field is accurate.

`0b1` The monitor instance is not ready and the contents of the MSMON_MBWU_L.VALUE field might be inaccurate or otherwise not represent the actual memory bandwidth usage.

Bits [62:44]

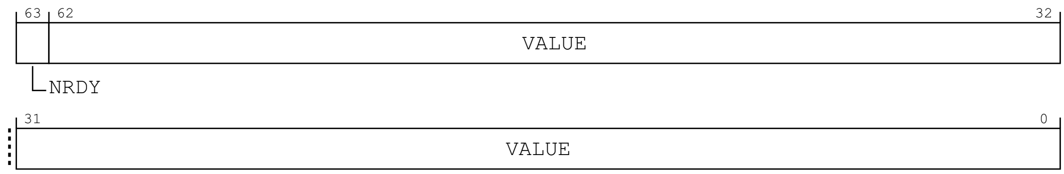
Reserved, RES0.

VALUE, bits [43:0]

Long (44-bit) memory bandwidth usage counter value if MSMON_MBWU_L.NRDY is 0. Invalid if MSMON_MBWU_L.NRDY is 1.

VALUE is the long count of bytes transferred since the monitor was last reset that met the criteria set in MSMON_CFG_MBWU_FLT and MSMON_CFG_MBWU_CTL for the monitor instance selected by MSMON_CFG_MON_SEL.

When MPAMF_MBWUMON_IDR.LWD == 1:



NRDY, bit [63]

Not Ready. Indicates whether the monitor instance has possibly inaccurate data.

`0b0` The monitor instance is ready and the MSMON_MBWU_L.VALUE field is accurate.

`0b1` The monitor instance is not ready and the contents of the MSMON_MBWU_L.VALUE field might be inaccurate or otherwise not represent the actual memory bandwidth usage.

VALUE, bits [62:0]

Long (63-bit) memory bandwidth usage counter value if MSMON_MBWU_L.NRDY is 0. Invalid if MSMON_MBWU_L.NRDY is 1.

VALUE is the long count of bytes transferred since the monitor instance was last reset that met the criteria set in MSMON_CFG_MBWU_FLT and MSMON_CFG_MBWU_CTL for the monitor instance selected by MSMON_CFG_MON_SEL.

Accessing the MSMON_MBWU_L:

This register is within the MPAM feature page memory frames.

In a system that supports Secure, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps.

- MSMON_MBWU_L_s must be accessible from the Secure MPAM feature page.
- MSMON_MBWU_L_ns must be accessible from the Non-secure MPAM feature page.
- MSMON_MBWU_L_rt must be accessible from the Root MPAM feature page.
- MSMON_MBWU_L_rl must be accessible from the Realm MPAM feature page.

MSMON_MBWU_L_s, MSMON_MBWU_L_ns, MSMON_MBWU_L_rt, and MSMON_MBWU_L_rl must be separate registers.

- The Secure instance (MSMON_MBWU_L_s) accesses the long memory bandwidth usage monitor used for Secure PARTIDs.
- The Non-secure instance (MSMON_MBWU_L_ns) accesses the long memory bandwidth usage monitor used for Non-secure PARTIDs.
- The Root instance (MSMON_MBWU_L_rt) accesses the long memory bandwidth usage monitor used for Root PARTIDs.
- The Realm instance (MSMON_MBWU_L_rl) accesses the long memory bandwidth usage monitor used for Realm PARTIDs.

When RIS is implemented, reads and writes to MSMON_MBWU_L access the long memory bandwidth usage monitor instance for the bandwidth resource instance selected by MSMON_CFG_MON_SEL.RIS and the monitor instance selected by MSMON_CFG_MON_SEL.MON_SEL.

When RIS is not implemented, reads and writes to MSMON_MBWU_L access the long memory bandwidth usage monitor instance for the monitor instance selected by MSMON_CFG_MON_SEL.MON_SEL.

MSMON_MBWU_L can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_s	0x0880	MSMON_MBWU_s

This interface is accessible as follows:

- Accesses to this register are RW.

MSMON_MBWU_L can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_ns	0x0880	MSMON_MBWU_ns

This interface is accessible as follows:

- Accesses to this register are RW.

MSMON_MBWU_L can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rt	0x0880	MSMON_MBWU_rt

This interface is accessible as follows:

- When FEAT_RME is implemented accesses to this register are RW.

MSMON_MBWU_L can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rl	0x0880	MSMON_MBWU_rl

This interface is accessible as follows:

- When FEAT_RME is implemented accesses to this register are RW.

11.5.13 MSMON_MBWU_L_CAPTURE, MPAM Long Memory Bandwidth Usage Monitor Capture Register

The MSMON_MBWU_L_CAPTURE characteristics are:

Purpose

Accesses the captured MSMON_MBWU_L monitor instance selected by MSMON_CFG_MON_SEL.

MSMON_MBWU_L_CAPTURE_s is the Secure long memory bandwidth usage monitor capture instance selected by the Secure instance of MSMON_CFG_MON_SEL.
MSMON_MBWU_L_CAPTURE_ns is the Non-secure long memory bandwidth usage monitor capture instance selected by the Non-secure instance of MSMON_CFG_MON_SEL.
MSMON_MBWU_L_CAPTURE_rt is the Root long memory bandwidth usage monitor capture instance selected by the Root instance of MSMON_CFG_MON_SEL.
MSMON_MBWU_L_CAPTURE_rl is the Realm long memory bandwidth usage monitor capture instance selected by the Realm instance of MSMON_CFG_MON_SEL.

If MPAMF_IDR.HAS_RIS is 1, the monitor instance long capture register accessed is for the resource instance currently selected by MSMON_CFG_MON_SEL.RIS and the monitor instance of that resource instance selected by MSMON_CFG_MON_SEL.MON_SEL.

Configurations

The power domain of MSMON_MBWU_L_CAPTURE is IMPLEMENTATION DEFINED.

This register is present only when FEAT_MPAM is implemented, MPAMF_IDR.HAS_MSMON == 1, MPAMF_MSMON_IDR.MSMON_MBWU == 1, MPAMF_MBWUMON_IDR.HAS_CAPTURE == 1 and MPAMF_MBWUMON_IDR.HAS_LONG == 1. Otherwise, direct accesses to MSMON_MBWU_L_CAPTURE are RES0.

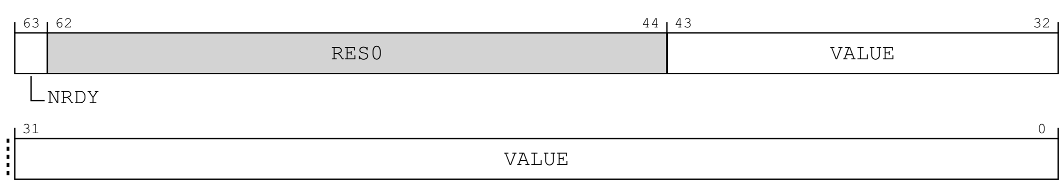
The power and reset domain of each MSC component is specific to that component.

Attributes

MSMON_MBWU_L_CAPTURE is a 64-bit register.

Field descriptions

When MPAMF_MBWUMON_IDR.LWD == 0:



NRDY, bit [63]

Not Ready. Indicates whether the monitor has possibly inaccurate data.

`0b0` The captured monitor instance was ready and the MSMON_MBWU_L_CAPTURE.VALUE field is accurate.

`0b1` The captured monitor instance was not ready and the contents of the MSMON_MBWU_L_CAPTURE.VALUE field might be inaccurate or otherwise not represent the actual memory bandwidth usage.

Bits [62:44]

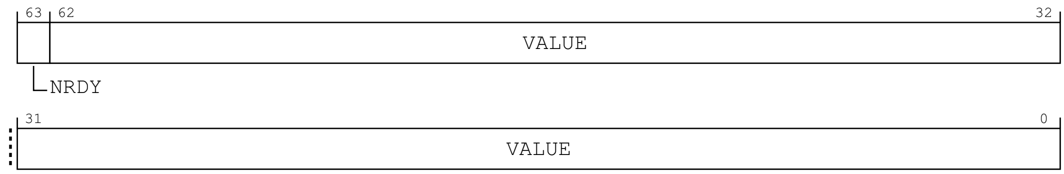
Reserved, RES0.

VALUE, bits [43:0]

Captured long memory bandwidth usage counter value if MSMON_MBWU_L_CAPTURE.NRDY is 0. Invalid if MSMON_MBWU_L_CAPTURE.NRDY is 1.

VALUE is the captured 44-bit count of bytes transferred since the monitor instance was last reset that met the criteria set in MSMON_CFG_MBWU_FLT and MSMON_CFG_MBWU_CTL for the monitor instance selected by MSMON_CFG_MON_SEL.

When MPAMF_MBWUMON_IDR.LWD == 1:



NRDY, bit [63]

Not Ready. Indicates whether the monitor has possibly inaccurate data.

`0b0` The captured monitor instance was ready and the MSMON_MBWU_L_CAPTURE.VALUE field is accurate.

`0b1` The captured monitor instance was not ready and the contents of the MSMON_MBWU_L_CAPTURE.VALUE field might be inaccurate or otherwise not represent the actual memory bandwidth usage.

VALUE, bits [62:0]

The captured long memory bandwidth usage counter value if MSMON_MBWU_L_CAPTURE.NRDY is 0. Invalid if MSMON_MBWU_L_CAPTURE.NRDY is 1.

VALUE is the captured 63-bit count of bytes transferred since the monitor instance was last reset that met the criteria set in MSMON_CFG_MBWU_FLT and MSMON_CFG_MBWU_CTL for the monitor instance selected by MSMON_CFG_MON_SEL.

Accessing the MSMON_MBWU_L_CAPTURE:

This register is within the MPAM feature page memory frames.

In a system that supports Secure, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps.

- MSMON_MBWU_L_CAPTURE_s must be accessible from the Secure MPAM feature page.
- MSMON_MBWU_L_CAPTURE_ns must be accessible from the Non-secure MPAM feature page.
- MSMON_MBWU_L_CAPTURE_rt must be accessible from the Root MPAM feature page.
- MSMON_MBWU_L_CAPTURE_rl must be accessible from the Realm MPAM feature page.

MSMON_MBWU_L_CAPTURE_s, MSMON_MBWU_L_CAPTURE_ns, MSMON_MBWU_L_CAPTURE_rt, and MSMON_MBWU_L_CAPTURE_rl must be separate registers.

- The Secure instance (MSMON_MBWU_L_CAPTURE_s) accesses the captured long memory bandwidth usage monitor used for Secure PARTIDs.
- The Non-secure instance (MSMON_MBWU_L_CAPTURE_ns) accesses the captured long memory bandwidth usage monitor used for Non-secure PARTIDs.
- The Root instance (MSMON_MBWU_L_CAPTURE_rt) accesses the captured long memory bandwidth usage monitor used for Root PARTIDs.

- The Realm instance (MSMON_MBWU_L_CAPTURE_rl) accesses the captured long memory bandwidth usage monitor used for Realm PARTIDs.

When RIS is implemented, reads and writes to MSMON_MBWU_L_CAPTURE access the monitor instance for the bandwidth resource instance selected by MSMON_CFG_MON_SEL.RIS and the memory bandwidth usage monitor instance selected by MSMON_CFG_MON_SEL.MON_SEL.

When RIS is not implemented, reads and writes to MSMON_MBWU_L_CAPTURE access the monitor instance for the memory bandwidth usage monitor instance selected by MSMON_CFG_MON_SEL.MON_SEL.

MSMON_MBWU_L_CAPTURE can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_s	`0x0890`	MSMON_MBWU_CAPTURE_s

This interface is accessible as follows:

- Accesses to this register are RW.

MSMON_MBWU_L_CAPTURE can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_ns	`0x0890`	MSMON_MBWU_CAPTURE_ns

This interface is accessible as follows:

- Accesses to this register are RW.

MSMON_MBWU_L_CAPTURE can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rt	`0x0890`	MSMON_MBWU_CAPTURE_rt

This interface is accessible as follows:

- When FEAT_RME is implemented accesses to this register are RW.

MSMON_MBWU_L_CAPTURE can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rl	`0x0890`	MSMON_MBWU_CAPTURE_rl

This interface is accessible as follows:

- When FEAT_RME is implemented accesses to this register are RW.

11.5.14 MSMON_MBWU_OFSR, MPAM MBWU Monitor Overflow Status Register

The MSMON_MBWU_OFSR characteristics are:

Purpose

MSMON_MBWU_OFSR is a 32-bit read-only register that shows bitmap of MBWU monitor instance overflow status for a contiguous group of 32 monitor instances.

MSMON_MBWU_OFSR_s gives a bitmap of pending MBWU overflow status for 32 Secure MBWU monitor instances. MSMON_MBWU_OFSR_ns gives a bitmap of pending MBWU overflow status for 32 Non-secure MBWU monitor instances. MSMON_MBWU_OFSR_rt gives a bitmap of pending MBWU overflow status for 32 Root MBWU monitor instances. MSMON_MBWU_OFSR_rl gives a bitmap of pending MBWU overflow status for 32 Realm MBWU monitor instances.

Configurations

The power domain of MSMON_MBWU_OFSR is IMPLEMENTATION DEFINED.

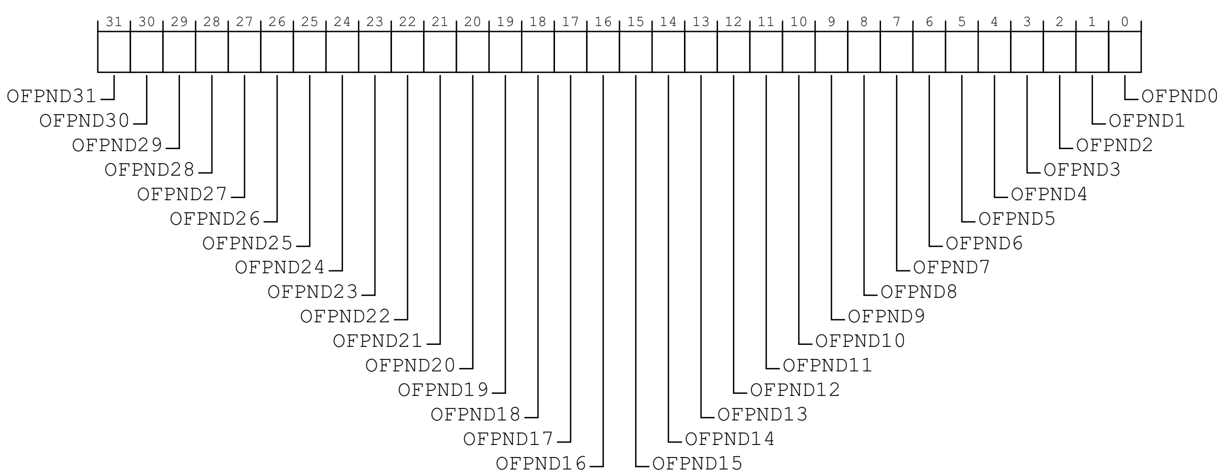
This register is present only when MPAMF_MBWUMON_IDR.HAS_OFSR == 1. Otherwise, direct accesses to MSMON_MBWU_OFSR are RES0.

The power and reset domain of each MSC component is specific to that component.

Attributes

MSMON_MBWU_OFSR is a 32-bit register.

Field descriptions



OFPND<i>, bit [i], for i = 31 to 0

Overflow status bitmap for MBWU monitor instances. The RIS and the contiguous range of MBWU monitor instances are set in MSMON_CFG_MON_SEL. i of 0 corresponds to the MBWU monitor instance MSMON_CFG_MON_SEL.MON_SEL & `0xFFE0`.

`0b0` MBWU monitor instance (MSMON_CFG_MON_SEL.MON_SEL & `0xFFE0` + i) does not have a pending overflow.

`0b1` MBWU monitor instance (MSMON_CFG_MON_SEL.MON_SEL & `0xFFE0` + i) has a pending overflow.

After reading MSMON_OFLOW_SR to determine that an MBWU monitor instance has a pending overflow and which RIS values have pending overflows, an interrupt service routine could poll groups of 32 monitor instances in a RIS for pending monitors by reading this bitmap and incrementing MSMON_CFG_MON_SEL.MON_SEL by 32.

A pending overflow may be in either the MSMON_CFG_MBWU_CTL.OFLOW_STATUS or MSMON_CFG_MBWU_CTL.OFLOW_STATUS_L field.

Accessing the MSMON_MBWU_OFSR:

This register is within the MPAM feature page memory frames.

In a system that supports Secure, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps.

- MSMON_MBWU_OFSR_s must be accessible from the Secure MPAM feature page.
- MSMON_MBWU_OFSR_ns must be accessible from the Non-secure MPAM feature page.
- MSMON_MBWU_OFSR_rt must be accessible from the Root MPAM feature page.
- MSMON_MBWU_OFSR_rl must be accessible from the Realm MPAM feature page.

MSMON_MBWU_OFSR_s, MSMON_MBWU_OFSR_ns, MSMON_MBWU_OFSR_rt, and MSMON_MBWU_OFSR_rl must be separate registers.

- The Secure instance (MSMON_MBWU_OFSR_s) accesses the MBWU monitor overflow status bitmap used for Secure PARTIDs.
- The Non-secure instance (MSMON_MBWU_OFSR_ns) accesses the MBWU monitor overflow status bitmap used for Non-secure PARTIDs.
- The Root instance (MSMON_MBWU_OFSR_rt) accesses the MBWU monitor overflow status bitmap used for Root PARTIDs.
- The Realm instance (MSMON_MBWU_OFSR_rl) accesses the MBWU monitor overflow status bitmap used for Realm PARTIDs.

MSMON_MBWU_OFSR can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_s	0x0898	MSMON_MBWU_OFSR_s

This interface is accessible as follows:

- Accesses to this register are RO.

MSMON_MBWU_OFSR can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_ns	0x0898	MSMON_MBWU_OFSR_ns

This interface is accessible as follows:

- Accesses to this register are RO.

MSMON_MBWU_OFSR can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rt	0x0898	MSMON_MBWU_OFSR_rt

This interface is accessible as follows:

- When FEAT_RME is implemented accesses to this register are RO.

MSMON_MBWU_OFSR can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rl	0x0898	MSMON_MBWU_OFSR_rl

This interface is accessible as follows:

- When FEAT_RME is implemented accesses to this register are RO.

11.5.15 MSMON_OFLOW_MSI_ADDR_H, MPAM Monitor Overflow MSI Write High-part Address Register

The MSMON_OFLOW_MSI_ADDR_H characteristics are:

Purpose

MSMON_OFLOW_MSI_ADDR_H is a 32-bit read/write register for the high part of the MPAM monitor overflow MSI address.

MSMON_OFLOW_MSI_ADDR_H_s is the high part of the MSI write address for monitor overflow interrupts from Secure monitor instances. MSMON_OFLOW_MSI_ADDR_H_ns is the high part of the MSI write address for monitor overflow interrupts from Non-secure monitor instances. MSMON_OFLOW_MSI_ADDR_H_rt is the high part of the MSI write address for monitor overflow interrupts from Root monitor instances. MSMON_OFLOW_MSI_ADDR_H_rl is the high part of the MSI write address for monitor overflow interrupts from Realm monitor instances.

Configurations

The power domain of MSMON_OFLOW_MSI_ADDR_H is IMPLEMENTATION DEFINED.

This register is present only when FEAT_MPAMv1p1 is implemented and MPAMF_MSMON_IDR.HAS_OFLW_MSI == 1. Otherwise, direct accesses to MSMON_OFLOW_MSI_ADDR_H are RES0.

MSMON_OFLOW_MSI_ADDR_L, MSMON_OFLOW_MSI_ADDR_H, MSMON_OFLOW_MSI_ATTR, MSMON_OFLOW_MSI_DATA, and MSMON_OFLOW_MSI_MPAM must all be implemented to support MSI writes for monitor overflow interrupts.

The power and reset domain of each MSC component is specific to that component.

Attributes

MSMON_OFLOW_MSI_ADDR_H is a 32-bit register.

Field descriptions

31 – 20	19 – 0
RES0	MSI_ADDR_H

Bits [31:20]

Reserved, RES0.

MSI_ADDR_H, bits [19:0]

MSI write address bits[51:32].

Accessing the MSMON_OFLOW_MSI_ADDR_H:

This register is within the MPAM feature page memory frames.

In a system that supports Secure, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps.

- MSMON_OFLW_MSI_ADDR_H_s must be accessible from the Secure MPAM feature page.
- MSMON_OFLW_MSI_ADDR_H_ns must be accessible from the Non-secure MPAM feature page.
- MSMON_OFLW_MSI_ADDR_H_rt must be accessible from the Root MPAM feature page.
- MSMON_OFLW_MSI_ADDR_H_rl must be accessible from the Realm MPAM feature page.

MSMON_OFLW_MSI_ADDR_H_s, MSMON_OFLW_MSI_ADDR_H_ns, MSMON_OFLW_MSI_ADDR_H_rt, and MSMON_OFLW_MSI_ADDR_H_rl must be separate registers.

- The Secure instance (MSMON_OFLW_MSI_ADDR_H_s) accesses the high part of the monitor overflow MSI write address of Secure monitors.
- The Non-secure instance (MSMON_OFLW_MSI_ADDR_H_ns) accesses the high part of the monitor overflow MSI write address of Non-secure monitors.
- The Root instance (MSMON_OFLW_MSI_ADDR_H_rt) accesses the high part of the monitor overflow MSI write address of Root monitors.
- The Realm instance (MSMON_OFLW_MSI_ADDR_H_rl) accesses the high part of the monitor overflow MSI write address of Realm monitors.

MSMON_OFLOW_MSI_ADDR_H can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_s	0x08E4	MSMON_OFLW_MSI_ADDR_H_s

This interface is accessible as follows:

- Accesses to this register are RW.

MSMON_OFLOW_MSI_ADDR_H can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_ns	0x08E4	MSMON_OFLW_MSI_ADDR_H_ns

This interface is accessible as follows:

- Accesses to this register are RW.

MSMON_OFLOW_MSI_ADDR_H can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rt	0x08E4	MSMON_OFLW_MSI_ADDR_H_rt

This interface is accessible as follows:

- When FEAT_RME is implemented accesses to this register are RW.

MSMON_OFLOW_MSI_ADDR_H can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rl	0x08E4	MSMON_OFLW_MSI_ADDR_H_rl

This interface is accessible as follows:

- When FEAT_RME is implemented accesses to this register are RW.

11.5.16 MSMON_OFLOW_MSI_ADDR_L, MPAM Monitor Overflow MSI Low-part Address Register

The MSMON_OFLOW_MSI_ADDR_L characteristics are:

Purpose

MSMON_OFLOW_MSI_ADDR_L is a 32-bit read/write register for the low part of the MPAM monitor MSI address.

MSMON_OFLOW_MSI_ADDR_L_s is the low part of the MSI write address for overflow interrupts from Secure monitor intances. MSMON_OFLOW_MSI_ADDR_L_ns is the low part of the MSI write address for overflow interrupts from Non-secure monitor instances. MSMON_OFLOW_MSI_ADDR_L_rt is the low part of the MSI write address for overflow interrupts from Root monitor intances. MSMON_OFLOW_MSI_ADDR_L_rl is the low part of the MSI write address for overflow interrupts from Realm monitor instances.

Configurations

The power domain of MSMON_OFLOW_MSI_ADDR_L is IMPLEMENTATION DEFINED.

This register is present only when FEAT_MPAMv1p1 is implemented and MPAMF_MSMON_IDR.HAS_OFLW_MSI == 1. Otherwise, direct accesses to MSMON_OFLOW_MSI_ADDR_L are RES0.

MSMON_OFLOW_MSI_ADDR_L, MSMON_OFLOW_MSI_ADDR_H, MSMON_OFLOW_MSI_ATTR, MSMON_OFLOW_MSI_DATA, and MSMON_OFLOW_MSI_MPAM must all be implemented to support MSI writes for monitor overflow interrupts.

The power and reset domain of each MSC component is specific to that component.

Attributes

MSMON_OFLOW_MSI_ADDR_L is a 32-bit register.

Field descriptions

31 … 2	1	0
MSI_ADDR_L	0	0

MSI_ADDR_L, bits [31:2]

MSI write address bits[31:2].

Bits [1:0]

Reads as `0b00`.

Access to this field is RO.

Accessing the MSMON_OFLOW_MSI_ADDR_L:

This register is within the MPAM feature page memory frames.

In a system that supports Secure, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps.

- MSMON_OFLOW_MSI_ADDR_L_s must be accessible from the Secure MPAM feature page.
- MSMON_OFLOW_MSI_ADDR_L_ns must be accessible from the Non-secure MPAM feature page.
- MSMON_OFLOW_MSI_ADDR_L_rt must be accessible from the Root MPAM feature page.
- MSMON_OFLOW_MSI_ADDR_L_rl must be accessible from the Realm MPAM feature page.

MSMON_OFLOW_MSI_ADDR_L_s, MSMON_OFLOW_MSI_ADDR_L_ns, MSMON_OFLOW_MSI_ADDR_L_rt, and MSMON_OFLOW_MSI_ADDR_L_rl must be separate registers.

- The Secure instance (MSMON_OFLOW_MSI_ADDR_L_s) accesses the low part of the overflow MSI write address used for Secure PARTIDs.
- The Non-secure instance (MSMON_OFLOW_MSI_ADDR_L_ns) accesses the low part of the overflow MSI write address used for Non-secure PARTIDs.
- The Root instance (MSMON_OFLOW_MSI_ADDR_L_rt) accesses the low part of the overflow MSI write address used for Root PARTIDs.
- The Realm instance (MSMON_OFLOW_MSI_ADDR_L_rl) accesses the low part of the overflow MSI write address used for Realm PARTIDs.

MSMON_OFLOW_MSI_ADDR_L can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_s	`0x08E0`	MSMON_OFLOW_MSI_ADDR_L_s

This interface is accessible as follows:

- Accesses to this register are RW.

MSMON_OFLOW_MSI_ADDR_L can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_ns	`0x08E0`	MSMON_OFLOW_MSI_ADDR_L_ns

This interface is accessible as follows:

- Accesses to this register are RW.

MSMON_OFLOW_MSI_ADDR_L can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rt	`0x08E0`	MSMON_OFLOW_MSI_ADDR_L_rt

This interface is accessible as follows:

- When FEAT_RME is implemented accesses to this register are RW.

MSMON_OFLOW_MSI_ADDR_L can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rl	`0x08E0`	MSMON_OFLOW_MSI_ADDR_L_rl

This interface is accessible as follows:

- When FEAT_RME is implemented accesses to this register are RW.

11.5.17 MSMON_OFLOW_MSI_ATTR, MPAM Monitor Overflow MSI Write Attributes Register

The MSMON_OFLOW_MSI_ATTR characteristics are:

Purpose

MSMON_OFLOW_MSI_ATTR is a 32-bit read/write register that controls MPAM monitor overflow MSI write attributes for MPAM monitor overflows in this MSC.

MSMON_OFLOW_MSI_ATTR_s controls Secure MPAM monitor overflow MSI writes.
MSMON_OFLOW_MSI_ATTR_ns controls Non-secure MPAM monitor overflow MSI writes.
MSMON_OFLOW_MSI_ATTR_rt controls Root MPAM monitor overflow MSI writes.
MSMON_OFLOW_MSI_ATTR_rl controls Realm MPAM monitor overflow MSI writes.

Configurations

The power domain of MSMON_OFLOW_MSI_ATTR is IMPLEMENTATION DEFINED.

This register is present only when FEAT_MPAMv1p1 is implemented and MPAMF_MSMON_IDR.HAS_OFLW_MSI == 1. Otherwise, direct accesses to MSMON_OFLOW_MSI_ATTR are RES0.

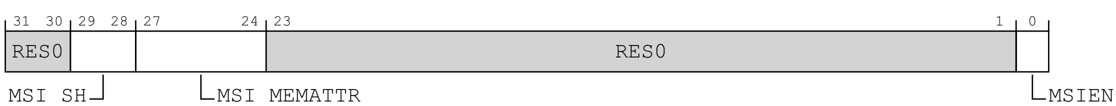
MSMON_OFLOW_MSI_ADDR_L, MSMON_OFLOW_MSI_ADDR_H, MSMON_OFLOW_MSI_ATTR, MSMON_OFLOW_MSI_DATA, and MSMON_OFLOW_MSI_MPAM must all be implemented to support MSI writes for monitor overflow interrupts.

The power and reset domain of each MSC component is specific to that component.

Attributes

MSMON_OFLOW_MSI_ATTR is a 32-bit register.

Field descriptions



Bits [31:30]

Reserved, RES0.

MSI_SH, bits [29:28]

Sharability attribute of MSI writes.

`0b00` Non-shareable.

`0b01` Reserved, CONSTRAINED UNPREDICTABLE.

`0b10` Outer Shareable.

`0b11` Inner Shareable.

When MSMON_OFLOW_MSI_ATTR.MSI_MEMATTR specifies a Device memory type, the contents of this field are IGNORED and Shareability is effectively Outer Shareable.

MSI_MEMATTR, bits [27:24]

Memory attributes of MSI writes.

Note: This encoding matches the VMSAv8-64 stage 2 MemAttr[3:0] field as described in the Arm ARM, except that the following encodings are Reserved (not UNPREDICTABLE) and behave as DEvice-nGnRnE: `0b0100`, `0b1000`, and `0b1100`.

`0b0000` Device-nGnRnE.

`0b0001` Device-nGnRE.

`0b0010` Device-nGRE.

`0b0011`	Device-GRE.
`0b0100`	Reserved. Behave as Device-nGnRnE, `0b0000`.
`0b0101`	Normal Inner Non-cacheable, Outer Non-cacheable.
`0b0110`	Normal Inner Write-Through Cacheable, Outer Non-cacheable.
`0b0111`	Normal Inner Write-Back Cacheable, Outer Non-cacheable.
`0b1000`	Reserved. Behave as Device-nGnRnE, `0b0000`.
`0b1001`	Normal Inner Non-Cachable, Outer Write-Through Cacheable.
`0b1010`	Normal Inner Write-Through Cacheable, Outer Write-Through Cachable.
`0b1011`	Normal Inner Write-Back Cacheable, Outer Write-Through Cachable.
`0b1100`	Reserved. Behave as Device-nGnRnE, `0b0000`.
`0b1101`	Normal Inner Non-cacheable, Outer Write-Back Cacheable.
`0b1110`	Normal Inner Write-Through Cacheable, Outer Write-Back Cacheable.
`0b1111`	Normal Inner Write-Back Cacheable, Outer Write-Back Cacheable.

When this field specifies a Device memory type, the contents of MSMON_OFLOW_MSI_ATTR.MSI_SH are IGNORED and Shareability is effectively Outer Shareable.

Device types may be implemented as any Device type with more n characters. For example, if this field is set to `0b0010`, an implementation may treat the MSI write as the specified type, Device-nGRE, or as Device-nGnRE or as Device-nGnRnE.

Reserved encodings `0b0100`, `0b1000`, and `0b1100` must be implemented to behave the same as the `0b0000` encoding.

Bits [23:1]

Reserved, RES0.

MSIEN, bit [0]

Monitor overflow MSI write enable.

`0b0`	MPAM monitor overflow MSI writes are not generated to signal enabled MPAM monitor overflow interrupts. When monitor overflow MSI writes are disabled, hardwired monitor overflow interrupt could be generated if hardwired monitor overflow interrupt is implemented.
`0b1`	MPAM monitor overflow MSI writes are generated to signal enabled MPAM monitor overflow interrupts. When monitor overflow MSI writes are enabled, hardwired monitor overflow interrupts are not generated.

This enable affects whether a hardwired overlow interrupt is generated.

The reset behavior of this field is:

- On a MSC reset, this field resets to `0`.

Accessing the MSMON_OFLOW_MSI_ATTR:

This register is within the MPAM feature page memory frames.

In a system that supports Secure, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps.

- MSMON_OFLOW_MSI_ATTR_s must be accessible from the Secure MPAM feature page.
- MSMON_OFLOW_MSI_ATTR_ns must be accessible from the Non-secure MPAM feature page.
- MSMON_OFLOW_MSI_ATTR_rt must be accessible from the Root MPAM feature page.
- MSMON_OFLOW_MSI_ATTR_rl must be accessible from the Realm MPAM feature page.

MSMON_OFLOW_MSI_ATTR_s, MSMON_OFLOW_MSI_ATTR_ns, MSMON_OFLOW_MSI_ATTR_rt, and MSMON_OFLOW_MSI_ATTR_rl must be separate registers.

- The Secure instance (MSMON_OFLOW_MSI_ATTR_s) accesses the monitor overflow MSI write attributes of Secure monitors.
- The Non-secure instance (MSMON_OFLOW_MSI_ATTR_ns) accesses the monitor overflow MSI write attributes of Non-secure monitors.
- The Root instance (MSMON_OFLOW_MSI_ATTR_rt) accesses the monitor overflow MSI write attributes of Root monitors.
- The Realm instance (MSMON_OFLOW_MSI_ATTR_rl) accesses the monitor overflow MSI write attributes of Realm monitors.

MSMON_OFLOW_MSI_ATTR can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_s	0x08EC	MSMON_OFLOW_MSI_ATTR_s

This interface is accessible as follows:

- Accesses to this register are RW.

MSMON_OFLOW_MSI_ATTR can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_ns	0x08EC	MSMON_OFLOW_MSI_ATTR_ns

This interface is accessible as follows:

- Accesses to this register are RW.

MSMON_OFLOW_MSI_ATTR can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rt	0x08EC	MSMON_OFLOW_MSI_ATTR_rt

This interface is accessible as follows:

- When FEAT_RME is implemented accesses to this register are RW.

MSMON_OFLOW_MSI_ATTR can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rl	0x08EC	MSMON_OFLOW_MSI_ATTR_rl

This interface is accessible as follows:

- When FEAT_RME is implemented accesses to this register are RW.

11.5.18 MSMON_OFLOW_MSI_DATA, MPAM Monitor Overflow MSI Write Data Register

The MSMON_OFLOW_MSI_DATA characteristics are:

Purpose

MSMON_OFLOW_MSI_DATA is a 32-bit read/write register for the MPAM monitor overflow MSI data.

MSMON_OFLOW_MSI_DATA_s is the data for the MSI write for monitor overflow from Secure monitor instances. MSMON_OFLOW_MSI_DATA_ns is the data for the MSI writes for monitor overflow interrupts from Non-secure monitor instances. MSMON_OFLOW_MSI_DATA_rt is the data for the MSI write for monitor overflow from Root monitor instances. MSMON_OFLOW_MSI_DATA_rl is the data for the MSI writes for monitor overflow interrupts from Realm monitor instances.

Configurations

The power domain of MSMON_OFLOW_MSI_DATA is IMPLEMENTATION DEFINED.

This register is present only when FEAT_MPAMv1p1 is implemented and MPAMF_MSMON_IDR.HAS_OFLW_MSI == 1. Otherwise, direct accesses to MSMON_OFLOW_MSI_DATA are RES0.

MSMON_OFLOW_MSI_ADDR_L, MSMON_OFLOW_MSI_ADDR_H, MSMON_OFLOW_MSI_ATTR, MSMON_OFLOW_MSI_DATA, and MSMON_OFLOW_MSI_MPAM must all be implemented to support MSI writes for monitor overflow interrupts.

The power and reset domain of each MSC component is specific to that component.

Attributes

MSMON_OFLOW_MSI_DATA is a 32-bit register.

Field descriptions

31 – 0
MSI_DATA

MSI_DATA, bits [31:0]

MSI write data word.

Accessing the MSMON_OFLOW_MSI_DATA:

This register is within the MPAM feature page memory frames.

In a system that supports Secure, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps.

- MSMON_OFLOW_MSI_DATA_s must be accessible from the Secure MPAM feature page.
- MSMON_OFLOW_MSI_DATA_ns must be accessible from the Non-secure MPAM feature page.
- MSMON_OFLOW_MSI_DATA_rt must be accessible from the Root MPAM feature page.
- MSMON_OFLOW_MSI_DATA_rl must be accessible from the Realm MPAM feature page.

MSMON_OFLOW_MSI_DATA_s, MSMON_OFLOW_MSI_DATA_ns, MSMON_OFLOW_MSI_DATA_rt, and MSMON_OFLOW_MSI_DATA_rl must be separate registers.

- The Secure instance (MSMON_OFLOW_MSI_DATA_s) accesses the monitor overflow MSI write data of Secure monitors.

- The Non-secure instance (MSMON_OFLOW_MSI_DATA_ns) accesses the monitor overflow MSI write data of Non-secure monitors.
- The Root instance (MSMON_OFLOW_MSI_DATA_rt) accesses the monitor overflow MSI write data of Root monitors.
- The Realm instance (MSMON_OFLOW_MSI_DATA_rl) accesses the monitor overflow MSI write data of Realm monitors.

MSMON_OFLOW_MSI_DATA can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_s	0x08E8	MSMON_OFLOW_MSI_DATA_s

This interface is accessible as follows:

- Accesses to this register are RW.

MSMON_OFLOW_MSI_DATA can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_ns	0x08E8	MSMON_OFLOW_MSI_DATA_ns

This interface is accessible as follows:

- Accesses to this register are RW.

MSMON_OFLOW_MSI_DATA can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rt	0x08E8	MSMON_OFLOW_MSI_DATA_rt

This interface is accessible as follows:

- When FEAT_RME is implemented accesses to this register are RW.

MSMON_OFLOW_MSI_DATA can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rl	0x08E8	MSMON_OFLOW_MSI_DATA_rl

This interface is accessible as follows:

- When FEAT_RME is implemented accesses to this register are RW.

11.5.19 MSMON_OFLOW_MSI_MPAM, MPAM Monitor Overflow MSI Write MPAM Information Register

The MSMON_OFLOW_MSI_MPAM characteristics are:

Purpose

MSMON_OFLOW_MSI_MPAM is a 32-bit read/write register that sets the MPAM information for a monitor overflow MSI write.

MSMON_OFLOW_MSI_MPAM_s controls MPAM information labeling of Secure monitor overflow MSI writes. MSMON_OFLOW_MSI_MPAM_ns controls MPAM information labeling of Non-secure monitor overflow MSI writes. MSMON_OFLOW_MSI_MPAM_rt controls MPAM information labeling of Root monitor overflow MSI writes. MSMON_OFLOW_MSI_MPAM_rl controls MPAM information labeling of Realm monitor overflow MSI writes.

Configurations

The power domain of MSMON_OFLOW_MSI_MPAM is IMPLEMENTATION DEFINED.

This register is present only when FEAT_MPAMv1p1 is implemented and MPAMF_MSMON_IDR.HAS_OFLW_MSI == 1. Otherwise, direct accesses to MSMON_OFLOW_MSI_MPAM are RES0.

MSMON_OFLOW_MSI_ADDR_L, MSMON_OFLOW_MSI_ADDR_H, MSMON_OFLOW_MSI_ATTR, MSMON_OFLOW_MSI_DATA, and MSMON_OFLOW_MSI_MPAM must all be implemented to support MSI writes for monitor overflow interrupts.

The power and reset domain of each MSC component is specific to that component.

Attributes

MSMON_OFLOW_MSI_MPAM is a 32-bit register.

Field descriptions

31–24	23–16	15–0
RES0	PMG	PARTID

Bits [31:24]

Reserved, RES0.

PMG, bits [23:16]

Performance monitoring group property for an MSC monitor overflow MSI write.

The reset behavior of this field is:

- On a MSC reset, this field resets to an architecturally UNKNOWN value.

PARTID, bits [15:0]

Partition ID for an MSC monitor overflow MSI write.

The PARTID in this field is in the Secure PARTID space in the MSMON_OFLOW_MSI_MPAM_s instance and in the Non-secure PARTID space in the MSMON_OFLOW_MSI_MPAM_ns instance of this register.

The reset behavior of this field is:

- On a MSC reset, this field resets to an architecturally UNKNOWN value.

Accessing the MSMON_OFLOW_MSI_MPAM:

This register is within the MPAM feature page memory frames.

In a system that supports Secure, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps.

- MSMON_OFLOW_MSI_MPAM_s must be accessible from the Secure MPAM feature page.
- MSMON_OFLOW_MSI_MPAM_ns must be accessible from the Non-secure MPAM feature page.
- MSMON_OFLOW_MSI_MPAM_rt must be accessible from the Root MPAM feature page.
- MSMON_OFLOW_MSI_MPAM_rl must be accessible from the Realm MPAM feature page.

MSMON_OFLOW_MSI_MPAM_s, MSMON_OFLOW_MSI_MPAM_ns, MSMON_OFLOW_MSI_MPAM_rt, and MSMON_OFLOW_MSI_MPAM_rl must be separate registers.

- The Secure instance (MSMON_OFLOW_MSI_MPAM_s) accesses the monitor overflow MSI MPAM information of Secure monitors.
- The Non-secure instance (MSMON_OFLOW_MSI_MPAM_ns) accesses the monitor overflow MSI MPAM information of Non-secure monitors.
- The Root instance (MSMON_OFLOW_MSI_MPAM_rt) accesses the monitor overflow MSI MPAM information of Root monitors.
- The Realm instance (MSMON_OFLOW_MSI_MPAM_rl) accesses the monitor overflow MSI MPAM information of Realm monitors.

MSMON_OFLOW_MSI_MPAM can be accessed through its memory-mapped interface:

Component	**Frame**	**Offset**	**Instance**
MPAM	MPAMF_BASE_s	`0x08DC`	MSMON_OFLOW_MSI_MPAM_s

This interface is accessible as follows:

- Accesses to this register are RW.

MSMON_OFLOW_MSI_MPAM can be accessed through its memory-mapped interface:

Component	**Frame**	**Offset**	**Instance**
MPAM	MPAMF_BASE_ns	`0x08DC`	MSMON_OFLOW_MSI_MPAM_ns

This interface is accessible as follows:

- Accesses to this register are RW.

MSMON_OFLOW_MSI_MPAM can be accessed through its memory-mapped interface:

Component	**Frame**	**Offset**	**Instance**
MPAM	MPAMF_BASE_rt	`0x08DC`	MSMON_OFLOW_MSI_MPAM_rt

This interface is accessible as follows:

- When FEAT_RME is implemented accesses to this register are RW.

MSMON_OFLOW_MSI_MPAM can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rl	0x08DC	MSMON_OFLOW_MSI_MPAM_rl

This interface is accessible as follows:

- When FEAT_RME is implemented accesses to this register are RW.

11.5.20 MSMON_OFLOW_SR, MPAM Monitor Overflow Status Register

The MSMON_OFLOW_SR characteristics are:

Purpose

MSMON_OFLOW_SR is a 32-bit read-only register that shows MPAM monitor overflow status for this MSC.

MSMON_OFLOW_SR_s gives the status of overflows of Secure MPAM monitors.
MSMON_OFLOW_SR_ns gives the status of overflows of Non-secure MPAM monitors.
MSMON_OFLOW_SR_rt gives the status of overflows of Root MPAM monitors.
MSMON_OFLOW_SR_rl gives the status of overflows of Realm MPAM monitors.

Configurations

The power domain of MSMON_OFLOW_SR is IMPLEMENTATION DEFINED.

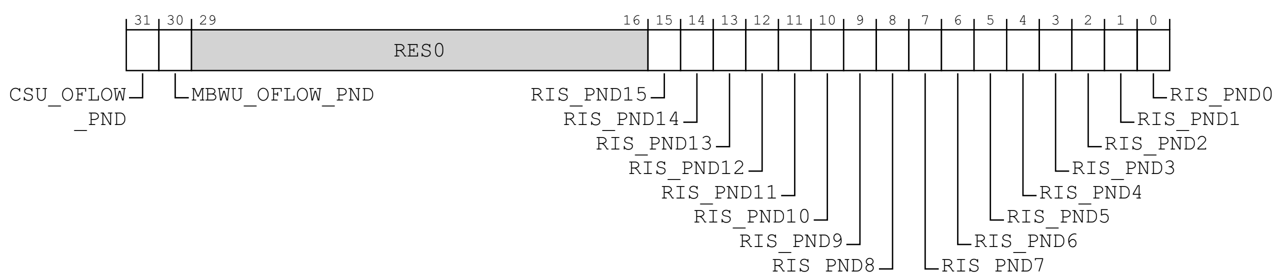
This register is present only when MPAMF_MSMON_IDR.HAS_OFLOW_SR == 1. Otherwise, direct accesses to MSMON_OFLOW_SR are RES0.

The power and reset domain of each MSC component is specific to that component.

Attributes

MSMON_OFLOW_SR is a 32-bit register.

Field descriptions



CSU_OFLOW_PND, bit [31]

At least one cache storage usage monitor has OFLOW_STATUS of 1 in MSMON_CFG_CSU_CTL.

`0b0` There are no cache storage usage monitor instances where MSMON_CFG_CSU_CTL.OFLOW_STATUS is 1.

`0b1` MSMON_CFG_CSU_CTL for at least one of the cache storage usage monitor instances has OFLOW_STATUS set to 1.

This field clears when MSMON_CFG_CSU_CTL.OFLOW_STATUS has been reset to 0 for all CSU monitor instances in this MSC.

MBWU_OFLOW_PND, bit [30]

At least one memory bandwidth usage monitor instance has OFLOW_STATUS or OFLOW_STATUS_L of 1 in MSMON_CFG_MBWU_CTL.

`0b0` There are no memory bandwidth usage monitor instances where MSMON_CFG_MBWU_CTL.OFLOW_STATUS is 1.

`0b1` MSMON_CFG_MBWU_CTL for at least one of the memory bandwidth usage monitor instances has either OFLOW_STATUS or OFLOW_STATUS_L set to 1.

This field clears when MSMON_CFG_MBWU_CTL.OFLOW_STATUS and MSMON_CFG_MBWU_CTL.OFLOW_STATUS_L have been reset to 0 for all MBWU monitor instances in this MSC.

Bits [29:16]

Reserved, RES0.

RIS_PND<r>, bit [r], for r = 15 to 0

Overflow status by RIS.

`0b0` RIS r has no unread overflows of any type of monitor.

`0b1` RIS r has at least one unread overflow in at least one of the monitor types.

Combined with the CSU_OFLOW_PND and MBWU_OFLOW_PND flags in this register, an interrupt service routine could poll only the monitor types indicated in monitors for the resource instances flagged in this field.

Bit r is set when any monitor instance of any type in resource instance r has OFLOW_STATUS or OFLOW_STATUS_L set to 1.

Accessing the MSMON_OFLOW_SR:

This register is within the MPAM feature page memory frames.

In a system that supports Secure, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps.

- MSMON_OFLOW_SR_s must be accessible from the Secure MPAM feature page.
- MSMON_OFLOW_SR_ns must be accessible from the Non-secure MPAM feature page.
- MSMON_OFLOW_SR_rt must be accessible from the Root MPAM feature page.
- MSMON_OFLOW_SR_rl must be accessible from the Realm MPAM feature page.

MSMON_OFLOW_SR_s, MSMON_OFLOW_SR_ns, MSMON_OFLOW_SR_rt, and MSMON_OFLOW_SR_rl must be separate registers.

- The Secure instance (MSMON_OFLOW_SR_s) accesses the monitor overflow status summary of Secure monitors.
- The Non-secure instance (MSMON_OFLOW_SR_ns) accesses the monitor overflow status summary of Non-secure monitors.
- The Root instance (MSMON_OFLOW_SR_rt) accesses the monitor overflow status summary of Root monitors.
- The Realm instance (MSMON_OFLOW_SR_rl) accesses the monitor overflow status summary of Realm monitors.

MSMON_OFLOW_SR can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_s	`0x08F0`	MSMON_OFLOW_SR_s

This interface is accessible as follows:

- Accesses to this register are RO.

MSMON_OFLOW_SR can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_ns	`0x08F0`	MSMON_OFLOW_SR_ns

This interface is accessible as follows:

- Accesses to this register are RO.

MSMON_OFLOW_SR can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rt	0x08F0	MSMON_OFLOW_SR_rt

This interface is accessible as follows:

- When FEAT_RME is implemented accesses to this register are RO.

MSMON_OFLOW_SR can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rl	0x08F0	MSMON_OFLOW_SR_rl

This interface is accessible as follows:

- When FEAT_RME is implemented accesses to this register are RO.

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. MPAMF_ECR_rt controls Root MPAM error handling. MPAMF_ECR_rl controls Realm MPAM error handling.

Configurations

The power domain of MPAMF_ECR is IMPLEMENTATION DEFINED.

This register is present only when FEAT_MPAM is implemented. Otherwise, direct accesses to MPAMF_ECR are RES0.

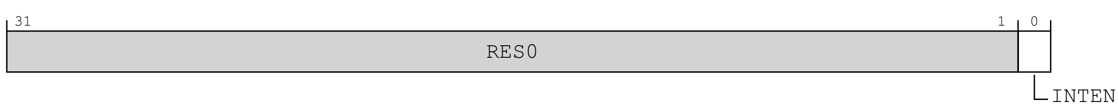
If an MSC cannot encounter any of the error conditions listed in Errors in MSCs, both the MPAMF_ESR and MPAMF_ECR must be RAZ/WI.

The power and reset domain of each MSC component is specific to that component.

Attributes

MPAMF_ECR is a 32-bit register.

Field descriptions



Bits [31:1]

Reserved, RES0.

INTEN, bit [0]

Interrupt Enable.

`0b0` MPAM error interrupts are not signaled.

`0b1` MPAM error interrupts are signaled.

Accessing the MPAMF_ECR:

This register is within the MPAM feature page memory frames.

In a system that supports Secure, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps.

- 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_rt must be accessible from the Root MPAM feature page.
- MPAMF_ECR_rl must be accessible from the Realm MPAM feature page.

MPAMF_ECR_s, MPAMF_ECR_ns, MPAMF_ECR_rt, and MPAMF_ECR_rl must be separate registers.

- The Secure instance (MPAMF_ECR_s) accesses the error interrupt controls used for Secure PARTIDs.
- The Non-secure instance (MPAMF_ECR_ns) accesses the error interrupt controls used for Non-secure PARTIDs.

- The Root instance (MPAMF_ECR_rt) accesses the error interrupt controls used for Root PARTIDs.
- The Realm instance (MPAMF_ECR_rl) accesses the error interrupt controls used for Realm 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.

MPAMF_ECR can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rt	0x00F0	MPAMF_ECR_rt

This interface is accessible as follows:

- When FEAT_RME is implemented accesses to this register are RW.

MPAMF_ECR can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rl	0x00F0	MPAMF_ECR_rl

This interface is accessible as follows:

- When FEAT_RME is implemented accesses to this register are RW.

11.6.2 MPAMF_ERR_MSI_ADDR_H, MPAM Error MSI High-part Address Register

The MPAMF_ERR_MSI_ADDR_H characteristics are:

Purpose

MPAMF_ERR_MSI_ADDR_H is a 32-bit read/write register for the high part of the MPAM error MSI address.

MPAMF_ERR_MSI_ADDR_H_s is the high part of the MSI write address for error interrupts related to Secure PARTIDs. MPAMF_ERR_MSI_ADDR_H_ns is the high part of the MSI write address for error interrupts related to Non-secure PARTIDs. MPAMF_ERR_MSI_ADDR_H_rt is the high part of the MSI write address for error interrupts related to Root PARTIDs. MPAMF_ERR_MSI_ADDR_H_rl is the high part of the MSI write address for error interrupts related to Realm PARTIDs.

Configurations

The power domain of MPAMF_ERR_MSI_ADDR_H is IMPLEMENTATION DEFINED.

This register is present only when MPAMF_IDR.HAS_ERR_MSI == 1. Otherwise, direct accesses to MPAMF_ERR_MSI_ADDR_H are RES0.

The power and reset domain of each MSC component is specific to that component.

Attributes

MPAMF_ERR_MSI_ADDR_H is a 32-bit register.

Field descriptions

31 : 20	19 : 0
RES0	MSI_ADDR_H

Bits [31:20]

Reserved, RES0.

MSI_ADDR_H, bits [19:0]

MSI write address bits[51:32].

Accessing the MPAMF_ERR_MSI_ADDR_H:

This register is within the MPAM feature page memory frames.

In a system that supports Secure, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps.

- MPAMF_ERR_MSI_ADDR_H_s must be accessible from the Secure MPAM feature page.
- MPAMF_ERR_MSI_ADDR_H_ns must be accessible from the Non-secure MPAM feature page.
- MPAMF_ERR_MSI_ADDR_H_rt must be accessible from the Root MPAM feature page.
- MPAMF_ERR_MSI_ADDR_H_rl must be accessible from the Realm MPAM feature page.

MPAMF_ERR_MSI_ADDR_H_s, MPAMF_ERR_MSI_ADDR_H_ns, MPAMF_ERR_MSI_ADDR_H_rt, and MPAMF_ERR_MSI_ADDR_H_rl must be separate registers.

- The Secure instance (MPAMF_ERR_MSI_ADDR_H_s) accesses the high part of the memory address for MSI write to signal an MPAM error used for Secure PARTIDs.
- The Non-secure instance (MPAMF_ERR_MSI_ADDR_H_ns) accesses the high part of the memory address for MSI write to signal an MPAM error used for Non-secure PARTIDs.

- The Root instance (MPAMF_ERR_MSI_ADDR_H_rt) accesses the high part of the memory address for MSI write to signal an MPAM error used for Root PARTIDs.
- The Realm instance (MPAMF_ERR_MSI_ADDR_H_rl) accesses the high part of the memory address for MSI write to signal an MPAM error used for Realm PARTIDs.

MPAMF_ERR_MSI_ADDR_H can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_s	0x00E4	MPAMF_ERR_MSI_ADDR_H_s

This interface is accessible as follows:

- Accesses to this register are RW.

MPAMF_ERR_MSI_ADDR_H can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_ns	0x00E4	MPAMF_ERR_MSI_ADDR_H_ns

This interface is accessible as follows:

- Accesses to this register are RW.

MPAMF_ERR_MSI_ADDR_H can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rt	0x00E4	MPAMF_ERR_MSI_ADDR_H_rt

This interface is accessible as follows:

- When FEAT_RME is implemented accesses to this register are RW.

MPAMF_ERR_MSI_ADDR_H can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rl	0x00E4	MPAMF_ERR_MSI_ADDR_H_rl

This interface is accessible as follows:

- When FEAT_RME is implemented accesses to this register are RW.

11.6.3 MPAMF_ERR_MSI_ADDR_L, MPAM Error MSI Low-part Address Register

The MPAMF_ERR_MSI_ADDR_L characteristics are:

Purpose

MPAMF_ERR_MSI_ADDR_L is a 32-bit read/write register for the low part of the MPAM error MSI address.

MPAMF_ERR_MSI_ADDR_L_s is the low part of the MSI write address for error interrupts related to Secure PARTIDs. MPAMF_ERR_MSI_ADDR_L_ns is the low part of the MSI write address for error interrupts related to Non-secure PARTIDs. MPAMF_ERR_MSI_ADDR_L_rt is the low part of the MSI write address for error interrupts related to Root PARTIDs. MPAMF_ERR_MSI_ADDR_L_rl is the low part of the MSI write address for error interrupts related to Realm PARTIDs.

Configurations

The power domain of MPAMF_ERR_MSI_ADDR_L is IMPLEMENTATION DEFINED.

This register is present only when MPAMF_IDR.HAS_ERR_MSI == 1. Otherwise, direct accesses to MPAMF_ERR_MSI_ADDR_L are RES0.

The power and reset domain of each MSC component is specific to that component.

Attributes

MPAMF_ERR_MSI_ADDR_L is a 32-bit register.

Field descriptions

31 … 2	1	0
MSI_ADDR_L	0	0

MSI_ADDR_L, bits [31:2]

MSI write address bits[31:2].

Bits [1:0]

Reads as `0b00`.

Access to this field is RO.

Accessing the MPAMF_ERR_MSI_ADDR_L:

This register is within the MPAM feature page memory frames.

In a system that supports Secure, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps.

- MPAMF_ERR_MSI_ADDR_L_s must be accessible from the Secure MPAM feature page.
- MPAMF_ERR_MSI_ADDR_L_ns must be accessible from the Non-secure MPAM feature page.
- MPAMF_ERR_MSI_ADDR_L_rt must be accessible from the Root MPAM feature page.
- MPAMF_ERR_MSI_ADDR_L_rl must be accessible from the Realm MPAM feature page.

MPAMF_ERR_MSI_ADDR_L_s, MPAMF_ERR_MSI_ADDR_L_ns, MPAMF_ERR_MSI_ADDR_L_rt, and MPAMF_ERR_MSI_ADDR_L_rl must be separate registers.

- The Secure instance (MPAMF_ERR_MSI_ADDR_L_s) accesses the low part of the memory address for MSI write to signal an MPAM error used for Secure PARTIDs.

- The Non-secure instance (MPAMF_ERR_MSI_ADDR_L_ns) accesses the low part of the memory address for MSI write to signal an MPAM error used for Non-secure PARTIDs.
- The Root instance (MPAMF_ERR_MSI_ADDR_L_rt) accesses the low part of the memory address for MSI write to signal an MPAM error used for Root PARTIDs.
- The Realm instance (MPAMF_ERR_MSI_ADDR_L_rl) accesses the low part of the memory address for MSI write to signal an MPAM error used for Realm PARTIDs.

MPAMF_ERR_MSI_ADDR_L can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_s	0x00E0	MPAMF_ERR_MSI_ADDR_L_s

This interface is accessible as follows:

- Accesses to this register are RW.

MPAMF_ERR_MSI_ADDR_L can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_ns	0x00E0	MPAMF_ERR_MSI_ADDR_L_ns

This interface is accessible as follows:

- Accesses to this register are RW.

MPAMF_ERR_MSI_ADDR_L can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rt	0x00E0	MPAMF_ERR_MSI_ADDR_L_rt

This interface is accessible as follows:

- When FEAT_RME is implemented accesses to this register are RW.

MPAMF_ERR_MSI_ADDR_L can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rl	0x00E0	MPAMF_ERR_MSI_ADDR_L_rl

This interface is accessible as follows:

- When FEAT_RME is implemented accesses to this register are RW.

11.6.4 MPAMF_ERR_MSI_ATTR, MPAM Error MSI Write Attributes Register

The MPAMF_ERR_MSI_ATTR characteristics are:

Purpose

MPAMF_ERR_MSI_ATTR is a 32-bit read/write register that controls MPAM error MSI write attributes for MPAM errors in this MSC.

MPAMF_ERR_MSI_ATTR_s controls the attributes of Secure MPAM error MSI writes.
MPAMF_ERR_MSI_ATTR_ns controls the attributes of Non-secure MPAM error MSI writes.
MPAMF_ERR_MSI_ATTR_rt controls the attributes of Root MPAM error MSI writes.
MPAMF_ERR_MSI_ATTR_rl controls the attributes of Realm MPAM error MSI writes.

Configurations

The power domain of MPAMF_ERR_MSI_ATTR is IMPLEMENTATION DEFINED.

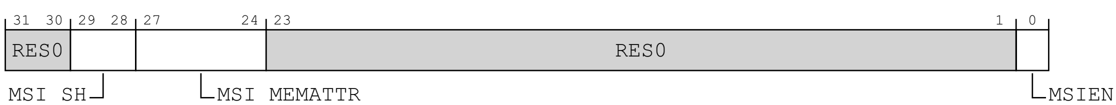
This register is present only when MPAMF_IDR.HAS_ERR_MSI == 1. Otherwise, direct accesses to MPAMF_ERR_MSI_ATTR are RES0.

The power and reset domain of each MSC component is specific to that component.

Attributes

MPAMF_ERR_MSI_ATTR is a 32-bit register.

Field descriptions



Bits [31:30]

Reserved, RES0.

MSI_SH, bits [29:28]

Sharability attribute of MSI writes.

`0b00` Non-shareable.

`0b01` Reserved, CONSTRAINED UNPREDICTABLE.

`0b10` Outer Shareable.

`0b11` Inner Shareable.

When MPAMF_ERR_MSI_ATTR.MSI_MEMATTR specifies a Device memory type, the contents of this field are IGNORED and Shareability is effectively Outer Shareable.

MSI_MEMATTR, bits [27:24]

Memory attributes of MSI writes.

Note: This encoding matches the VMSAv8-64 stage 2 MemAttr[3:0] field as described in the Arm ARM, except that the following encodings are Reserved (not UNPREDICTABLE) and behave as DEvice-nGnRnE: `0b0100`, `0b1000`, and `0b1100`.

`0b0000` Device-nGnRnE.

`0b0001` Device-nGnRE.

`0b0010` Device-nGRE.

`0b0011` Device-GRE.

`0b0100` Reserved. Behave as Device-nGnRnE, `0b0000`.

`0b0101` Normal Inner Non-cacheable, Outer Non-cacheable.

0b0110	Normal Inner Write-Through Cacheable, Outer Non-cacheable.
0b0111	Normal Inner Write-Back Cacheable, Outer Non-cacheable.
0b1000	Reserved. Behave as Device-nGnRnE, 0b0000.
0b1001	Normal Inner Non-Cachable, Outer Write-Through Cacheable.
0b1010	Normal Inner Write-Through Cacheable, Outer Write-Through Cacheable.
0b1011	Normal Inner Write-Back Cacheable, Outer Write-Through Cacheable.
0b1100	Reserved. Behave as Device-nGnRnE, 0b0000.
0b1101	Normal Inner Non-cacheable, Outer Write-Back Cacheable.
0b1110	Normal Inner Write-Through Cacheable, Outer Write-Back Cacheable.
0b1111	Normal Inner Write-Back Cacheable, Outer Write-Back Cacheable.

When this field specifies a Device memory type, the contents of MPAMF_ERR_MSI_ATTR.MSI_SH are IGNORED and Shareability is effectively Outer Shareable.

Device types may be implemented as any Device type with more than 'n' characters. For example, if this field is set to 0b0010, an implementation may treat the MSI write as the specified type, Device-nGRE, or as Device-nGnRE or as Device-nGnRnE.

Reserved encodings 0b0100, 0b1000, and 0b1100 must be implemented to behave the same as the 0b0000 encoding.

Bits [23:1]

Reserved, RES0.

MSIEN, bit [0]

Error interrupt MSI Enable.

0b0	MPAM error MSI writes are not generated to signal enabled MPAM error interrupts. When error MSI writesare disabled, hardwired error interrupts could be generated.
0b1	MPAM error MSI writes are generated to signal enabled MPAM error interrupts. When error MSI writes are enabled, hardwired error interrupts are not generated.

The value of this field affects whether hardwired error interrupts are generated.

The reset behavior of this field is:

- On a MSC reset, this field resets to 0.

Accessing the MPAMF_ERR_MSI_ATTR:

This register is within the MPAM feature page memory frames.

In a system that supports Secure, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps.

- MPAMF_ERR_MSI_ATTR_s must be accessible from the Secure MPAM feature page.
- MPAMF_ERR_MSI_ATTR_ns must be accessible from the Non-secure MPAM feature page.
- MPAMF_ERR_MSI_ATTR_rt must be accessible from the Root MPAM feature page.
- MPAMF_ERR_MSI_ATTR_rl must be accessible from the Realm MPAM feature page.

MPAMF_ERR_MSI_ATTR_s, MPAMF_ERR_MSI_ATTR_ns, MPAMF_ERR_MSI_ATTR_rt, and MPAMF_ERR_MSI_ATTR_rl must be separate registers.

- The Secure instance (MPAMF_ERR_MSI_ATTR_s) accesses the memory access attributes for MSI write to signal an MPAM error used for Secure PARTIDs.
- The Non-secure instance (MPAMF_ERR_MSI_ATTR_ns) accesses the memory access attributes for MSI write to signal an MPAM error used for Non-secure PARTIDs.

- The Root instance (MPAMF_ERR_MSI_ATTR_rt) accesses the memory access attributes for MSI write to signal an MPAM error used for Root PARTIDs.
- The Realm instance (MPAMF_ERR_MSI_ATTR_rl) accesses the memory access attributes for MSI write to signal an MPAM error used for Realm PARTIDs.

MPAMF_ERR_MSI_ATTR can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_s	0x00EC	MPAMF_ERR_MSI_ATTR_s

This interface is accessible as follows:

- Accesses to this register are RW.

MPAMF_ERR_MSI_ATTR can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_ns	0x00EC	MPAMF_ERR_MSI_ATTR_ns

This interface is accessible as follows:

- Accesses to this register are RW.

MPAMF_ERR_MSI_ATTR can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rt	0x00EC	MPAMF_ERR_MSI_ATTR_rt

This interface is accessible as follows:

- When FEAT_RME is implemented accesses to this register are RW.

MPAMF_ERR_MSI_ATTR can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rl	0x00EC	MPAMF_ERR_MSI_ATTR_rl

This interface is accessible as follows:

- When FEAT_RME is implemented accesses to this register are RW.

11.6.5 MPAMF_ERR_MSI_DATA, MPAM Error MSI Data Register

The MPAMF_ERR_MSI_DATA characteristics are:

Purpose

MPAMF_ERR_MSI_DATA is a 32-bit read/write register for the MPAM error MSI data.

MPAMF_ERR_MSI_DATA_s is the data for the MSI write for error interrupts related to Secure PARTIDs. MPAMF_ERR_MSI_DATA_ns is the data for the MSI write for error interrupts related to Non-secure PARTIDs. MPAMF_ERR_MSI_DATA_rt is the data for the MSI write for error interrupts related to Root PARTIDs. MPAMF_ERR_MSI_DATA_rl is the data for the MSI write for error interrupts related to Realm PARTIDs.

Configurations

The power domain of MPAMF_ERR_MSI_DATA is IMPLEMENTATION DEFINED.

This register is present only when MPAMF_IDR.HAS_ERR_MSI == 1. Otherwise, direct accesses to MPAMF_ERR_MSI_DATA are RES0.

The power and reset domain of each MSC component is specific to that component.

Attributes

MPAMF_ERR_MSI_DATA is a 32-bit register.

Field descriptions

31–0
MSI_DATA

MSI_DATA, bits [31:0]

MSI data to be written to ITS to signal an MSI.

Accessing the MPAMF_ERR_MSI_DATA:

This register is within the MPAM feature page memory frames.

In a system that supports Secure, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps.

- MPAMF_ERR_MSI_DATA_s must be accessible from the Secure MPAM feature page.
- MPAMF_ERR_MSI_DATA_ns must be accessible from the Non-secure MPAM feature page.
- MPAMF_ERR_MSI_DATA_rt must be accessible from the Root MPAM feature page.
- MPAMF_ERR_MSI_DATA_rl must be accessible from the Realm MPAM feature page.

MPAMF_ERR_MSI_DATA_s, MPAMF_ERR_MSI_DATA_ns, MPAMF_ERR_MSI_DATA_rt, and MPAMF_ERR_MSI_DATA_rl must be separate registers.

- The Secure instance (MPAMF_ERR_MSI_DATA_s) accesses the data for MSI write to signal an MPAM error used for Secure PARTIDs.
- The Non-secure instance (MPAMF_ERR_MSI_DATA_ns) accesses the data for MSI write to signal an MPAM error used for Non-secure PARTIDs.
- The Root instance (MPAMF_ERR_MSI_DATA_rt) accesses the data for MSI write to signal an MPAM error used for Root PARTIDs.
- The Realm instance (MPAMF_ERR_MSI_DATA_rl) accesses the data for MSI write to signal an MPAM error used for Realm PARTIDs.

MPAMF_ERR_MSI_DATA can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_s	0x00E8	MPAMF_ERR_MSI_DATA_s

This interface is accessible as follows:

- Accesses to this register are RW.

MPAMF_ERR_MSI_DATA can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_ns	0x00E8	MPAMF_ERR_MSI_DATA_ns

This interface is accessible as follows:

- Accesses to this register are RW.

MPAMF_ERR_MSI_DATA can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rt	0x00E8	MPAMF_ERR_MSI_DATA_rt

This interface is accessible as follows:

- When FEAT_RME is implemented accesses to this register are RW.

MPAMF_ERR_MSI_DATA can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rl	0x00E8	MPAMF_ERR_MSI_DATA_rl

This interface is accessible as follows:

- When FEAT_RME is implemented accesses to this register are RW.

11.6.6 MPAMF_ERR_MSI_MPAM, MPAM Error MSI Write MPAM Information Register

The MPAMF_ERR_MSI_MPAM characteristics are:

Purpose

MPAMF_ERR_MSI_MPAM is a 32-bit read/write register that sets the MPAM information for error MSI write attributes for MPAM errors in this MSC.

MPAMF_ERR_MSI_MPAM_s controls MPAM information labeling of Secure MPAM error MSI writes. MPAMF_ERR_MSI_MPAM_ns controls MPAM information labeling of Non-secure MPAM error MSI writes. MPAMF_ERR_MSI_MPAM_rt controls MPAM information labeling of Root MPAM error MSI writes. MPAMF_ERR_MSI_MPAM_rl controls MPAM information labeling of Realm MPAM error MSI writes.

Configurations

The power domain of MPAMF_ERR_MSI_MPAM is IMPLEMENTATION DEFINED.

This register is present only when MPAMF_IDR.HAS_ERR_MSI == 1. Otherwise, direct accesses to MPAMF_ERR_MSI_MPAM are RES0.

The power and reset domain of each MSC component is specific to that component.

Attributes

MPAMF_ERR_MSI_MPAM is a 32-bit register.

Field descriptions

31–24	23–16	15–0
RES0	PMG	PARTID

Bits [31:24]

Reserved, RES0.

PMG, bits [23:16]

Performance monitoring group property for PARTID MSC error interrupt write.

The reset behavior of this field is:

- On a MSC reset, this field resets to an architecturally UNKNOWN value.

PARTID, bits [15:0]

Partition ID for MSC error interrupt write.

The PARTID in this register is in the Secure PARTID space in the MPAMF_ERR_MSI_MPAM_s instance and in the Non-secure PARTID space in the MPAMF_ERR_MSI_MPAM_ns instance of this register.

The reset behavior of this field is:

- On a MSC reset, this field resets to an architecturally UNKNOWN value.

Accessing the MPAMF_ERR_MSI_MPAM:

This register is within the MPAM feature page memory frames.

In a system that supports Secure, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps.

- MPAMF_ERR_MSI_MPAM_s must be accessible from the Secure MPAM feature page.
- MPAMF_ERR_MSI_MPAM_ns must be accessible from the Non-secure MPAM feature page.

- MPAMF_ERR_MSI_MPAM_rt must be accessible from the Root MPAM feature page.
- MPAMF_ERR_MSI_MPAM_rl must be accessible from the Realm MPAM feature page.

MPAMF_ERR_MSI_MPAM_s, MPAMF_ERR_MSI_MPAM_ns, MPAMF_ERR_MSI_MPAM_rt, and MPAMF_ERR_MSI_MPAM_rl must be separate registers.

- The Secure instance (MPAMF_ERR_MSI_MPAM_s) accesses the MPAM information for MSI write request to signal an MPAM error used for Secure PARTIDs.
- The Non-secure instance (MPAMF_ERR_MSI_MPAM_ns) accesses the MPAM information for MSI write request to signal an MPAM error used for Non-secure PARTIDs.
- The Root instance (MPAMF_ERR_MSI_MPAM_rt) accesses the MPAM information for MSI write request to signal an MPAM error used for Root PARTIDs.
- The Realm instance (MPAMF_ERR_MSI_MPAM_rl) accesses the MPAM information for MSI write request to signal an MPAM error used for Realm PARTIDs.

MPAMF_ERR_MSI_MPAM can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_s	0x00DC	MPAMF_ERR_MSI_MPAM_s

This interface is accessible as follows:

- Accesses to this register are RW.

MPAMF_ERR_MSI_MPAM can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_ns	0x00DC	MPAMF_ERR_MSI_MPAM_ns

This interface is accessible as follows:

- Accesses to this register are RW.

MPAMF_ERR_MSI_MPAM can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rt	0x00DC	MPAMF_ERR_MSI_MPAM_rt

This interface is accessible as follows:

- When FEAT_RME is implemented accesses to this register are RW.

MPAMF_ERR_MSI_MPAM can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rl	0x00DC	MPAMF_ERR_MSI_MPAM_rl

This interface is accessible as follows:

- When FEAT_RME is implemented accesses to this register are RW.

11.6.7 MPAMF_ESR, MPAM Error Status Register

The MPAMF_ESR characteristics are:

Purpose

Indicates MPAM error status for this MSC.

MPAMF_ESR_s reports Secure MPAM errors. MPAMF_ESR_ns reports Non-secure MPAM errors. MPAMF_ESR_rt reports Root MPAM errors. MPAMF_ESR_rl reports Realm 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.

This register is present only when FEAT_MPAM is implemented. Otherwise, direct accesses to MPAMF_ESR are RES0.

MPAMF_ESR is 64-bit register when MPAM v0.1 or v1.1 is implemented and MPAMF_IDR.HAS_EXTD_ESR == 1.

Otherwise, MPAMF_ESR is a 32-bit register.

If an MSC cannot encounter any of the error conditions listed in Errors in MSCs, both the MPAMF_ESR and MPAMF_ECR must be RAZ/WI.

The power and reset domain of each MSC component is specific to that component.

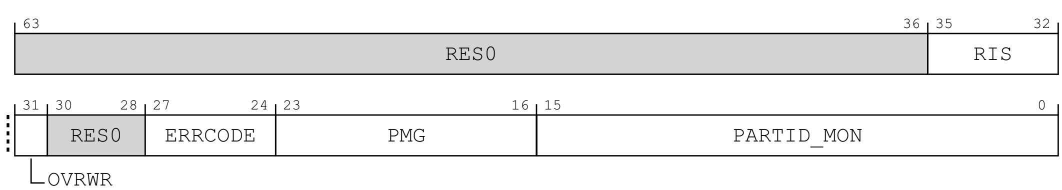
Attributes

MPAMF_ESR is a:

- 64-bit register when (FEAT_MPAMv0p1 is implemented or FEAT_MPAMv1p1 is implemented) and MPAMF_IDR.HAS_EXTD_ESR == 1
- 32-bit register otherwise

Field descriptions

When (FEAT_MPAMv0p1 is implemented or FEAT_MPAMv1p1 is implemented) and MPAMF_IDR.HAS_EXTD_ESR == 1:



Bits [63:36]

Reserved, RES0.

RIS, bits [35:32]

When MPAMF_IDR.HAS_RIS == 1:

Resource Instance Selector. Where applicable to the ERRCODE, captures the RIS value for the error.

Otherwise:

Reserved, RES0.

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.

`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`	Undefined_RIS_PART_SEL.
`0b1001`	RIS_No_Control.
`0b1010`	Undefined_RIS_MON_SEL.
`0b1011`	RIS_No_Monitor.
`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]

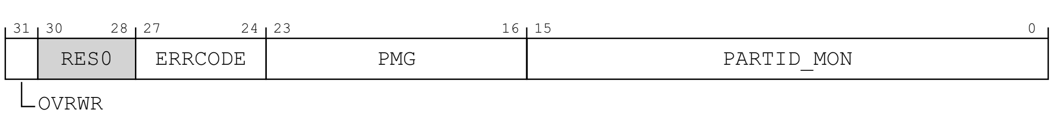
PARTID or monitor.

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 0.

Otherwise:



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 0 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.

`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, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps.

- 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_rt must be accessible from the Root MPAM feature page.
- MPAMF_ESR_rl must be accessible from the Realm MPAM feature page.

MPAMF_ESR_s, MPAMF_ESR_ns, MPAMF_ESR_rt, and MPAMF_ESR_rl must be separate registers.

- The Secure instance (MPAMF_ESR_s) accesses the error status used for Secure PARTIDs.
- The Non-secure instance (MPAMF_ESR_ns) accesses the error status used for Non-secure PARTIDs.
- The Root instance (MPAMF_ESR_rt) accesses the error status used for Root PARTIDs.
- The Realm instance (MPAMF_ESR_rl) accesses the error status used for Realm 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.

MPAMF_ESR can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rt	0x00F8	MPAMF_ESR_rt

This interface is accessible as follows:

- When FEAT_RME is implemented accesses to this register are RW.

MPAMF_ESR can be accessed through its memory-mapped interface:

Component	Frame	Offset	Instance
MPAM	MPAMF_BASE_rl	0x00F8	MPAMF_ESR_rl

This interface is accessible as follows:

- When FEAT_RME is implemented 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. See *Optionality of error detection and reporting* on page 12-375.

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 and does not detect whether the unimplemented high-order bits of the PARTID are all zero, 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 12-365 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 8-166. 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.

When Resource instance selection is implemented, hardware is permitted to make choices regarding CONSTRAINED UNPREDICTABLE behaviors and unimplemented RIS bits that could reduce or remove the need to detect or report any of the RIS-related errors. For more information on RIS, see *Resource instance selection* on page 8-158.

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 on page 12-365. Other fields are captured in MPAMF_ESR as shown in the "Fields Captured" column of Table 12-1 on page 12-365.

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 and RIS[a]
2	Req_PARTID_Range	A request has out-of-range PARTID.	PARTID, PMG
3	MSMONCFG_ID_RANGE	MSMON configuration request has out-of-range PARTID or PMG.	PARTID, PMG, RIS[a]
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, RIS[a]
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
8	Undefined_RIS_PART_SEL	Unimplemented RIS in MPAMCFG_PART_SEL.RIS.	PART_SEL, RIS
9	RIS_No_Control	Resource instance selected by MPAMCFG_PART_SEL.RIS does not have the accessed partitioning control.	PART_SEL, RIS
10	Undefined_RIS_MON_SEL	Unimplemented RIS in MSMON_CFG_MON_SEL.	MON_SEL, RIS
11	RIS_No_Monitor	Resource instance selected by MSMON_CFG_MON_SEL.RIS does not have the accessed monitor type.	MON_SEL, RIS
12:18	Reserved	Reserved for future use.	--

a. This field is only available when MPAMF_IDR.EXT and MPAMF_IDR.HAS_RIS are 1.

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 3-38.

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 3-38.
- The default PARTID and PMG may be used for the request's MPAM_NS Security state. See *Default PARTID* on page 3-38. 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_INTPARTIDINTERNAL == 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 Undefined RIS in MPAMCFG_PART_SEL.RIS (errorcode == 8)

This error occurs when an access to an MPAMCFG_* register occurs when MPAMCFG_PART_SEL.RIS does not correspond to a RIS value allocated to an MPAM resource of the MSC. The MPAM behavior of an MSC for a request that causes this error is a CONSTRAINED UNPREDICTABLE choice between:

- RAZ/WI.
- RAZ/WI and record an MPAM error in the MPAMF_ESR associated with that MSC, using the error code ERRCODE == 8 and capturing MPAMCFG_PART_SEL.{RIS, PARTID_SEL}.

12.2.10 RIS in MPAMCFG_PART_SEL.RIS does not have partitioning control (errorcode == 9)

This error occurs when an access to an MPAMCFG_* register occurs when MPAMCFG_PART_SEL.RIS selects a resource that exists but does not have the partitioning control accessed. The MPAM behavior of an MSC for a request that causes this error is a CONSTRAINED UNPREDICTABLE choice between:

- RAZ/WI.
- RAZ/WI and record an MPAM error in the MPAMF_ESR associated with that MSC, using the error code ERRCODE == 9 and capturing MPAMCFG_PART_SEL.{RIS, PARTID_SEL}.

12.2.11 Undefined RIS in MSMON_CFG_MON_SEL.RIS (errorcode == 10)

This error occurs when an access to an MSMON_CFG_* register occurs when MSMON_CFG_MON_SEL.RIS does not correspond to an MPAM resource of the MSC. The MPAM behavior of an MSC for a request that causes this error is a CONSTRAINED UNPREDICTABLE choice between:

- RAZ/WI.
- RAZ/WI and record an MPAM error in the MPAMF_ESR associated with that MSC, using the error code ERRCODE == 10 and capturing MSMON_CFG_MON_SEL.{RIS, MON_SEL}.

12.2.12 RIS selected by MSMON_CFG_MON_SEL.RIS does not have monitor type (errorcode == 11)

Access to an MSMON_<type> or MSMON_<type>_CAPTURE register when MSMON_CFG_MON_SEL.RIS does not correspond to an MPAM resource of the MSC or that does not have the type of monitor accessed by the MSMON_<type> or MSMON_<type>_CAPTURE register. The MPAM behavior of an MSC for a request that causes this error is a CONSTRAINED UNPREDICTABLE choice between:

- Read as `0xFFFFFFFE`, NRDY == 1 with value of `0x7FFFFFFE`, and WI. This value is highly unlikely as a normal return value in any monitor.
- RAZ/WI.
- RAZ/WI and record an MPAM error in the MPAMF_ESR associated with that MSC, using the error code ERRCODE == 11 and capturing MSMON_CFG_MON_SEL.{RIS, MON_SEL}.

Access to an MSMON_<type>_* register when MSMON_CFG_MON_SEL.RIS does not correspond to an MPAM resource that has the type of monitor accessed by the MSMON_<type> _* register is CONSTRAINED UNPREDICTABLE, one of:

- RAZ/WI.
- RAZ/WI and record an MPAM error in the MPAMF_ESR associated with that MSC, using the error code ERRCODE == 11 and capturing MSMON_CFG_MON_SEL.{RIS, MON_SEL}.

12.2.13 Reserved (errcodes 12 – 15)

These error codes are 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

12.4.1 Writing an out-of-range PARTID to MPAMCFG_PART_SEL.PARTID_SEL

If a write to MPAMCFG_PART_SEL has a PARTID_SEL value that is out-of-range, it is IMPLEMENTATION DEFINED whether:

- The contents written to MPAMCFG_PART_SEL.PARTID_SEL are not checked at the time of the write and store the new value into MPAMCFG_PART_SEL.PARTID_SEL. The written out-of-range value could later cause a PARTID_SEL out-of-range error (ERRCODE = 1) when used to index an access to another configuration register by PARTID_SEL. See *Required error condition detection* on page 12-375 for more information about the optionality of error detection.

- The contents being written to MPAMCFG_PART_SEL.PARTID_SEL are checked before updating the MPAMCFG_PART_SEL register. If the error is detected, the MPAMCFG_PART_SEL register is not updated and the PARTID_SEL out-of-range error (ERRCODE = 1) is raised. To implement this behavior, the implementation must detect the error.

12.4.2 Reading another MPAMCFG_* register when MPAMCFG_PART_SEL.PARTID_SEL contains an out-of-range PARTID

A read of any MPAMCFG_* register other than MPAMCFG_PART_SEL when MPAMCFG_PART_SEL.PARTID_SEL contains an out-of-range PARTID raises a PARTID_SEL out-of-range error (ERRCODE = 1) if that error is detected. See *Required error condition detection* on page 12-375 for more information about the optionality of error detection.

It is IMPLEMENTATION DEFINED whether the value returned by a read of another MPAMCFG_* register when MPAMCFG_PART_SEL.PARTID_SEL contains an out-of-range PARTID that is detected:

- Is an UNKNOWN value.
- Is a constant value of zero in all fields.

The value returned by a read of another MPAMCFG_* register when MPAMCFG_PART_SEL.PARTID_SEL contains an out-of-range PARTID that is not detected is an UNKNOWN value.

Note

In an implementation that chooses the IMPLEMENTATION DEFINED option to detect out-of-range PARTID_SEL values and to not update the MPAMCFG_PART_SEL register, it is not possible to have an out-of-range PARTID_SEL value in that register and the precondition for this section cannot occur. See *Writing an out-of-range PARTID to MPAMCFG_PART_SEL.PARTID_SEL* on page 12-370.

12.4.3 Writing another MPAMCFG_* register when MPAMCFG_PART_SEL.PARTID_SEL contains an out-of-range PARTID

A write of any MPAMCFG_* register other than MPAMCFG_PART_SEL when MPAMCFG_PART_SEL.PARTID_SEL contains an out-of-range PARTID raises a PARTID_SEL out-of-range error (ERRCODE = 1) if that error is detected. See *Required error condition detection* on page 12-375 for more information about the optionality of error detection.

If a write to an MPAMCFG_* register other than MPAMCFG_PART_SEL has a PARTID_SEL out-of-range error (ERRCODE = 1), whether that error is detected or not detected, it is IMPLEMENTATION DEFINED whether:

- The write updates the configuration register indexed by an UNKNOWN in-range PARTID.
- The write is ignored (WI).

Note

In an implementation that chooses the IMPLEMENTATION DEFINED option to detect out-of-range PARTID_SEL values and to not update the MPAMCFG_PART_SEL register, it is not possible to have an out-of-range PARTID_SEL value in that register and the precondition for this section cannot occur. See *Writing an out-of-range PARTID to MPAMCFG_PART_SEL.PARTID_SEL* on page 12-370.

12.4.4 Writing an undefined RIS to MPAMCFG_PART_SEL.RIS

If RIS is implemented and a configuration write to MPAMCFG_PART_SEL.RIS has an Undefined RIS error (ERRCODE = 8), it is IMPLEMENTATION DEFINED whether:

- The contents written to MPAMCFG_PART_SEL.RIS are not checked at the time of the write and store the new value in MPAMCFG_PART_SEL.RIS. This undefined RIS value could cause an Undefined RIS error (ERRCODE = 8) when later used to select a resource on an access to a configuration register by PARTID_SEL and RIS.
- The contents being written to MPAMCFG_PART_SEL.RIS are checked before updating the MPAMCFG_PART_SEL register. If the error is detected, the MPAMCFG_PART_SEL register is not updated and the Undefined RIS error (ERRCODE = 8) is raised. To implement this behavior, the implementation must detect the error.

12.4.5 Reading other MSC MPAM registers when MPAMCFG_PART_SEL.RIS contains an undefined RIS value

A read of an MPAMF*IDR register or an MPAMCFG_* register other than MPAMCFG_PART_SEL when MPAMCFG_PART_SEL.RIS contains an undefined RIS value raises an Undefined RIS error (ERRCODE = 8) if the implementation detects that error. See *Required error condition detection* on page 12-375 for more information about the optionality of error detection. If the error is not detected, the value returned is UNKNOWN.

The value read from an MPAMF*IDR or an MPAMCFG_* register other than MPAMCFG_PART_SEL when MPAMCFG_PART_SEL.RIS contains a RIS value that does not correspond to an implemented resource instance returns an UNKNOWN value.

Note

In an implementation that chooses the IMPLEMENTATION DEFINED option to detect undefined RIS values and to not update the MPAMCFG_PART_SEL register, it is not possible to have an undefined RIS value in that register and the precondition for this section cannot occur. See *Writing an undefined RIS to MPAMCFG_PART_SEL.RIS* on page 12-371.

12.4.6 Writing other MSC MPAM registers when MPAMCFG_PART_SEL.RIS contains an undefined RIS value

A write of an MPAMCFG_* register other than MPAMCFG_PART_SEL when MPAMCFG_PART_SEL.RIS contains an undefined RIS value raises an Undefined RIS error (ERRCODE = 8) if that error is detected. See *Required error condition detection* on page 12-375 for more information about the optionality of error detection.

If a configuration write to an MPAMCFG_* register other than MPAMCFG_PART_SEL has a RIS value that does not correspond to an implemented resource instance, whether the undefined RIS error is detected or not detected, it is IMPLEMENTATION DEFINED whether:

- The write might update the configuration register for any implemented resource instance.
- The write is ignored (WI).

Note

In an implementation that chooses the IMPLEMENTATION DEFINED option to detect undefined RIS values and to not update the MPAMCFG_PART_SEL register, it is not possible to have an undefined RIS value in that register and the precondition for this section cannot occur. See *Writing an undefined RIS to MPAMCFG_PART_SEL.RIS* on page 12-371.

12.4.7 Reads of MSC MPAM registers with other errors

If there is no PARTID_SEL out-of-range error (ERRCODE = 1) and no Undefined RIS error (ERRCODE = 8), a configuration read to an MPAM*IDR or an MPAMCFG_* register that has any other errors detected returns an UNKNOWN value.

12.4.8 Writes to MSC MPAM registers with other errors

If there is no PARTID_SEL out-of-range error (ERRCODE = 1) and no Undefined RIS error (ERRCODE = 8), a configuration write to an MPAMCFG_* register that has any other errors detected leaves the control settings for the partition selected by MPAMCFG_PART_SEL.PARTID_SEL and MPAMCFG_PART_SEL.RIS in an UNKNOWN state.

12.4.9 Writes to MSMON_CFG_MON_SEL.MON_SEL

Writes to MSMON_CFG_MON_SEL that have the MON_SEL field out-of-range for the monitors of the MSC cannot generally be detected when the MON_SEL register is written because different types of monitors could have different numbers of supported monitor instances. If RIS is also implemented, then the resource instance selector being written into the RIS field could change which monitor types are available and how many monitor instances of each type are implemented because different resource instances could have different numbers of monitor instances from the same resource type.

There are limited cases where MSMON_CFG_MON_SEL.MON_SEL could be checked when written:

- RIS is not implemented and only a single monitor type is supported.
- RIS is not supported and all supported monitor types have exactly the same number of monitor instances.
- RIS is supported and all monitor types of all resource instances support exactly the same number of monitor instances.
- RIS is supported, different resource instances support a different number of monitor instances, and all monitor types of each resource instance support exactly the same number of monitor instances. In this case the RIS value must be used to determine the maximum number of monitor instances to check the MON_SEL value.

Checking for out-of-range MON_SEL when MSMON_CFG_MON_SEL is written is an implementation option because some of the detectable cases could be common.

If a configuration write to MSMON_CFG_MON_SEL has a MON_SEL value that is out-of-range, it is IMPLEMENTATION DEFINED whether:

- The contents written to MSMON_CFG_MON_SEL.MON_SEL are not checked at the time of the write and store the new value into the register. The written out-of-range value could later cause a MON_SEL out-of-range error (ERRCODE = 5) when used to index an access to a MSMON_CFG_* configuration register or MSMON_* monitor or capture register by MON_SEL.
- The contents being written to MSMON_CFG_MON_SEL.MON_SEL are checked before updating the MSMON_CFG_MON_SEL register. If the error is detected, the MSMON_CFG_MON_SEL register is not updated and the MON_SEL out-of-range error (ERRCODE = 5) is raised. See *Required error condition detection* on page 12-375 for more information about the optionality of error detection.

12.4.10 Reading another MSMON_* register when MSMON_CFG_MON_SEL.MON_SEL out of range

A read of any MSMON_* register other than MSMON_CFG_MON_SEL when MSMON_CFG_MON_SEL.MON_SEL contains an out-of-range monitor instance selector raises a Monitor Range error (ERRCODE == 5) if that error is detected. See *Required error condition detection* on page 12-375 for more information about the optionality of error detection.

The value read from any MSMON_* register other than MSMON_CFG_MON_SEL when MSMON_CFG_MON_SEL.MON_SEL contains an out-of-range monitor instance selector returns an UNKNOWN value whether the Monitor Range error is detected or not detected.

Note

In an implementation that chooses the IMPLEMENTATION DEFINED option to detect out-of-range MON_SEL values and to not update the MSMON_CFG_MON_SEL register, it might not be possible to have an out-of-range MON_SEL value in that register and the precondition for this section cannot occur. Section *Writes to MSMON_CFG_MON_SEL.MON_SEL* on page 12-372 lists the conditions necessary to permit the choice of this option.

12.4.11 Writes to MSMON_* registers with MSMON_CFG_MON_SEL.MON_SEL out of range

A write of any MSMON_* register other than MSMON_CFG_MON_SEL when MSMON_CFG_MON_SEL.MON_SEL contains an out-of-range monitor instance selector, raises a Monitor Range error (ERRCODE == 5) if that error is detected. See *Required error condition detection* on page 12-375 for more information about the optionality of error detection.

If a write is to an MSMON_* register other than MSMON_CFG_MON_SEL when MSMON_CFG_MON_SEL.MON_SEL is out-of-range, whether the error is detected or not detected, it is IMPLEMENTATION DEFINED whether:

- The write could update an MSMON_* register indexed by any in-range monitor instance selector.
- The write is ignored (WI).

Note

In an implementation that chooses the IMPLEMENTATION DEFINED option to detect out-of-range MON_SEL values and to not update the MSMON_CFG_MON_SEL register, it might not be possible to have an out-of-range MON_SEL value in that register and the precondition for this section cannot occur. *Writes to MSMON_CFG_MON_SEL.MON_SEL* on page 12-372 lists the conditions necessary to permit the choice of this option.

12.4.12 Writing an undefined RIS to MSMON_CFG_MON_SEL.RIS

If RIS is implemented and a configuration write to MSMON_CFG_MON_SEL.RIS has a value that does not correspond to an implemented resource instance, it is IMPLEMENTATION DEFINED whether:

- The value written to MSMON_CFG_MON_SEL.RIS is not checked at the time of the write and the new values are stored in that register. This undefined RIS value could cause an Undefined_RIS_MON_SEL error (ERRCODE = 10) when later used to select a resource on an access to an MSMON_* register by MON_SEL and RIS.
- The contents being written to MSMON_CFG_MON_SEL.RIS are checked before updating the MSMON_CFG_MON_SEL register. If the error is detected, the register is not updated and the Undefined_RIS_MON_SEL error (ERRCODE = 10) is raised.

12.4.13 Reading another MSMON_* register when MSMON_CFG_MON_SEL.RIS contains an undefined RIS value

A read of an MSMON_* register other than MSMON_CFG_MON_SEL when MSMON_CFG_MON_SEL.RIS contains a RIS value that does not correspond to an implemented resource instance raises an Undefined_RIS_MON_SEL error (ERRCODE = 10) if that error is detected. See *Required error condition detection* on page 12-375 for more information about the optionality of error detection.

The value read from an MSMON_* register other than MSMON_CFG_MON_SEL when MSMON_CFG_MON_SEL.RIS contains a RIS value that does not correspond to an implemented resource instance returns an UNKNOWN value whether the error is detected or not detected.

Note

In an implementation that chooses the IMPLEMENTATION DEFINED option to detect undefined RIS values and to not update the MSMON_CFG_MON_SEL register, it is not possible to have an undefined RIS value in that register and the precondition for this section cannot occur. See *Writing an undefined RIS to MSMON_CFG_MON_SEL.RIS* on page 12-373.

12.4.14 Writing another MSMON_* register when MSMON_CFG_MON_SEL.RIS contains an undefined RIS value

A write of an MSMON_* register other than MSMON_CFG_MON_SEL when MSMON_CFG_MON_SEL.RIS contains a RIS value that does not correspond to an implemented resource instance raises an Undefined_RIS_MON_SEL error (ERRCODE = 10) if that error is detected. See *Required error condition detection* on page 12-375 for more information about the optionality of error detection.

If a write to an MSMON_* register other than MSMON_CFG_MON_SEL has a RIS value that does not correspond to an implemented resource, whether the undefined RIS error is detected or not detected, it is IMPLEMENTATION DEFINED whether:

- The write might update the MSMON_* register indexed by any implemented resource instance.
- The write is ignored (WI).

Note

In an implementation that chooses the IMPLEMENTATION DEFINED option to detect undefined RIS values and to not update the MSMON_CFG_MON_SEL register, it is not possible to have an undefined RIS value in that register and the precondition for this section cannot occur. See *Writing an undefined RIS to MSMON_CFG_MON_SEL.RIS* on page 12-373.

12.5 Optionality of error detection and reporting

Error detection and reporting are required for an error condition when all of the following are true:

- The MSC supports at least one MPAM feature that can raise the error condition.
- The MSC is designed so that the particular error condition can occur.
- The MSC is required to detect the error condition, see *Required error condition detection* on page 12-375.

If there are no error conditions that meet these criteria, then in MPAM v0.1 and from MPAM v1.1, MPAMF_IDR.HAS_ESR is permitted to be 0. If MPAMF_IDR.HAS_ESR is 1, then MPAMF_ESR and MPAMF_ECR must be implemented.

In MPAM v1.0, if no error conditions are detected, MPAMF_ESR and MPAMF_ECR must be RAZ/WI.

12.5.1 Required error condition detection

This section describes the conditions under which each of the MPAM MSC error conditions must be detected. In cases where detection is not required, an implementation might choose not to implement detection and reporting logic for that error condition.

Selector out-of-range errors

The following requirements apply to each of the types of selectors used in MPAM in MSCs, including:

- PARTID.
- PMG.
- Monitor selectors.
- In MPAM v0.1 and from MPAM v1.1, RIS values.

The selector interface is permitted to be narrower than the full width specified in the architecture. Even if the MSC interface is of one size, the internal implementation might be smaller than that size. Bits beyond the implemented width of any selector are permitted to be silently truncated without any requirement to detect or report should those bits be non-zero.

An MSC implementation that supports a range that is not 0 to 2^n -1 in a field of n bits for any selector is required to detect and report values that lie within the field size but are not valid in the implementation. Such detection can be applied after performing the silent truncation to the bit-width supported.

PARTID narrowing errors

If PARTID narrowing is supported, the Unexpected Internal error condition must be detected and reported.

Chapter 13
Pseudocode

This chapter contains pseudocode that describes the generation of MPAM information by a PE following the MPAM architecture. It contains the following section:

- *Shared pseudocode* on page 13-378.

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* on page 13-378.
- *shared/functions/memory* on page 13-378.
- *shared/functions/mpam* on page 13-379.

13.1.1 shared/functions/extension

This section includes the following pseudocode functions:

- *shared/functions/extension/HaveEMPAMExt* on page 13-378.
- *shared/functions/extension/HaveMPAMExt* on page 13-378.

shared/functions/extension/HaveEMPAMExt

```
// HaveEMPAMExt()
// ==============
// Returns TRUE if Enhanced MPAM is implemented, and FALSE otherwise.
boolean HaveEMPAMExt()
    return (HasArchVersion(ARMv8p6) && HaveMPAMExt() &&
            boolean IMPLEMENTATION_DEFINED "Has enhanced MPAM extension");
```

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* on page 13-378.
- *shared/functions/memory/CreateAccessDescriptor* on page 13-378.
- *shared/functions/memory/MPAM* on page 13-379.

shared/functions/memory/AccessDescriptor

```
type AccessDescriptor is (
    boolean transactional,
    MPAMinfo mpam,
    AccType acctype)
```

shared/functions/memory/CreateAccessDescriptor

```
// CreateAccessDescriptor()
// ========================

AccessDescriptor CreateAccessDescriptor(AccType acctype)
    AccessDescriptor accdesc;
    accdesc.acctype = acctype;
    accdesc.transactional = FALSE;
```

```
    accdesc.mpam = GenMPAMcurEL(acctype);
    return accdesc;
```

shared/functions/memory/MPAM

```
// MPAM Types
// ==========

type PARTIDtype = bits(16);
type PMGtype = bits(8);
type PARTIDspaceType = bits(2);
constant PARTIDspaceType PIdSpace_Secure    = '00';
constant PARTIDspaceType PIdSpace_NonSecure = '01';
constant PARTIDspaceType PIdSpace_Root      = '10';
constant PARTIDspaceType PIdSpace_Realm     = '11';

type MPAMinfo is (
     PARTIDspaceType mpam_sp,
     PARTIDtype partid,
     PMGtype pmg
)
```

13.1.3 shared/functions/mpam

This section includes the following pseudocode functions:

shared/functions/mpam/AltPARTIDspace

```
// AltPARTIDspace()
// ================
// From the Security state, EL and ALTSP configuration, determine
// whether to primary space or the alt space is selected and which
// PARTID space is the alternative space. Return that alternative
// PARTID space if selected or the primary space if not.

PARTIDspaceType AltPARTIDspace(bits(2) el, SecurityState security,
                                PARTIDspaceType primaryPIdSpace)
    case security of
        when SS_NonSecure
```

```
            assert el != EL3;
            return primaryPIdSpace; // there is no ALTSP for Non_secure
        when SS_Secure
            assert el != EL3;
            if primaryPIdSpace == PIdSpace_NonSecure then
                return primaryPIdSpace;
            return AltPIdSecure(el, primaryPIdSpace);
        when SS_Root
            assert el == EL3;
            if MPAM3_EL3.ALTSP_EL3 == '1' then
                if MPAM3_EL3.RT_ALTSP_NS == '1' then
                    return PIdSpace_NonSecure;
                else
                    return PIdSpace_Secure;
            else
                return primaryPIdSpace;
        when SS_Realm
            assert el != EL3;
            return AltPIdRealm(el, primaryPIdSpace);
        otherwise
            Unreachable();
```

shared/functions/mpam/AltPIdRealm

```
// AltPIdRealm()
// =============
// Compute PARTID space as either the primary PARTID space or
// alternative PARTID space in the Realm Security state.
// Helper for AltPARTIDspace.

PARTIDspaceType AltPIdRealm(bits(2) el, PARTIDspaceType primaryPIdSpace)
    PARTIDspaceType PIdSpace = primaryPIdSpace;
    case el of
        when EL0
            if ELIsInHost(EL0) then
                if !UsePrimarySpaceEL2() then
                    PIdSpace = PIdSpace_NonSecure;
            elseif !UsePrimarySpaceEL10() then
                PIdSpace = PIdSpace_NonSecure;
        when EL1
            if !UsePrimarySpaceEL10() then
                PIdSpace = PIdSpace_NonSecure;
        when EL2
            if !UsePrimarySpaceEL2() then
                PIdSpace = PIdSpace_NonSecure;
        otherwise
            Unreachable();
    return PIdSpace;
```

shared/functions/mpam/AltPIdSecure

```
// AltPIdSecure()
// ==============
// Compute PARTID space as either the primary PARTID space or
// alternative PARTID space in the Secure Security state.
// Helper for AltPARTIDspace.

PARTIDspaceType AltPIdSecure(bits(2) el, PARTIDspaceType primaryPIdSpace)
    PARTIDspaceType PIdSpace = primaryPIdSpace;
    boolean el2en = EL2Enabled();
      case el of
          when EL0
              if el2en then
                  if ELIsInHost(EL0) then
                      if !UsePrimarySpaceEL2() then
```

```
                    PIdSpace = PIdSpace_NonSecure;
                elsif !UsePrimarySpaceEL10() then
                    PIdSpace = PIdSpace_NonSecure;
            elsif MPAM3_EL3.ALTSP_HEN == '0' && MPAM3_EL3.ALTSP_HFC == '1' then
                PIdSpace = PIdSpace_NonSecure;
        when EL1
            if el2en then
                if !UsePrimarySpaceEL10() then
                    PIdSpace = PIdSpace_NonSecure;
            elsif MPAM3_EL3.ALTSP_HEN == '0' && MPAM3_EL3.ALTSP_HFC == '1' then
            PIdSpace = PIdSpace_NonSecure;
        when EL2
            if !UsePrimarySpaceEL2() then
                PIdSpace = PIdSpace_NonSecure;
        otherwise
            Unreachable();
    return PIdSpace;
```

shared/functions/mpam/DefaultMPAMinfo

```
// DefaultMPAMinfo()
// =================
// Returns default MPAM info.  The partidspace argument sets
// the PARTID space of the default MPAM information returned.

MPAMinfo DefaultMPAMinfo(PARTIDspaceType partidspace)
    MPAMinfo DefaultInfo;
    DefaultInfo.mpam_sp = partidspace;
    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.
// 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(AccType acctype)
    bits(2) mpamEL;
    boolean validEL = FALSE;
    SecurityState security = AArch64.CurrentSecurityState();
    boolean InD = FALSE;
    PARTIDspaceType pspace = PARTIDspaceFromSS(security);
    if pspace == PIdSpace_NonSecure && !MPAMisEnabled() then
        return DefaultMPAMinfo(pspace);
    if UsingAArch32() then
        (validEL, mpamEL) = ELFromM32(PSTATE.M);
    else
        mpamEL = PSTATE.EL;
        validEL = TRUE;
```

```
    case acctype of
        when AccType_IFETCH, AccType_IC
            InD = TRUE;
        otherwise
            // other access types are DATA accesses
            InD = FALSE;
    if !validEL then
        return DefaultMPAMinfo(pspace);
    elsif MPAMIDR_EL1.HAS_ALTSP == '1' then
        // Substitute alternative PARTID space if selected
        pspace = AltPARTIDspace(mpamEL, security, pspace);
    if HaveEMPAMExt() && security == SS_Secure then
        if MPAM3_EL3.FORCE_NS == '1' then
            pspace = PIdSpace_NonSecure;
        if MPAM3_EL3.SDEFLT == '1' then
            return DefaultMPAMinfo(pspace);
    if !MPAMisEnabled() then
        return DefaultMPAMinfo(pspace);
    else
        return genMPAM(mpamEL, InD, pspace);
```

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 vpmrmax = UInt(MPAMIDR_EL1.VPMR_MAX);

    // vpartid_max is largest vpartid supported
    integer vpartid_max = (vpmrmax << 2) + 3;

    // One of many ways to reduce vpartid to value less than vpartid_max.
    if UInt(vpartid) > 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(bits(2) el)
    return (MPAMIDR_EL1.HAS_HCR == '1' && EL2Enabled() &&
            ((el == EL0 && MPAMHCR_EL2.EL0_VPMEN == '1' &&
               (HCR_EL2.E2H == '0' || HCR_EL2.TGE == '0')) ||
             (el == EL1 && MPAMHCR_EL2.EL1_VPMEN == '1')));
```

shared/functions/mpam/PARTIDspaceFromSS

```
// PARTIDspaceFromSS()
// ===================
// Returns the primary PARTID space from the Security State.

PARTIDspaceType PARTIDspaceFromSS(SecurityState security)
    case security of
        when SS_NonSecure
            return PIdSpace_NonSecure;
        when SS_Root
            return PIdSpace_Root;
        when SS_Realm
            return PIdSpace_Realm;
        when SS_Secure
            return PIdSpace_Secure;
        otherwise
            Unreachable();
```

shared/functions/mpam/UsePrimarySpaceEL10

```
// UsePrimarySpaceEL10()
// =====================
// Checks whether Primary space is configured in the
// MPAM3_EL3 and MPAM2_EL2 ALTSP control bits that affect
// MPAM ALTSP use at EL1 and EL0.

boolean UsePrimarySpaceEL10()
    boolean hen3 = MPAM3_EL3.ALTSP_HEN == '1';
    return ((!hen3 && MPAM3_EL3.ALTSP_HFC == '0') ||
            ( hen3 && MPAM2_EL2.ALTSP_HFC == '0'));
```

shared/functions/mpam/UsePrimarySpaceEL2

```
// UsePrimarySpaceEL2()
// ====================
// Checks whether Primary space is configured in the
// MPAM3_EL3 and MPAM2_EL2 ALTSP control bits that affect
// MPAM ALTSP use at EL2.

boolean UsePrimarySpaceEL2()
    boolean hen3 = MPAM3_EL3.ALTSP_HEN == '1';
    return ((!hen3 && MPAM3_EL3.ALTSP_HFC == '0') ||
            ( hen3 && MPAM2_EL2.ALTSP_EL2 == '0'));
```

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 PARTID in PARTID space pspace.

MPAMinfo genMPAM(bits(2) el, boolean InD, PARTIDspaceType pspace)
    MPAMinfo returninfo;
    PARTIDtype partidel;
    boolean perr;
    // gstplk is guest OS application locked by the EL2 hypervisor to
    // only use EL1 the virtual machine's PARTIDs.
    boolean gstplk = (el == EL0 && EL2Enabled() &&
                      MPAMHCR_EL2.GSTAPP_PLK == '1' &&
                      HCR_EL2.TGE == '0');
    bits(2) eff_el = if gstplk then EL1 else el;
    (partidel, perr) = genPARTID(eff_el, InD);
    PMGtype groupel  = genPMG(eff_el, InD, perr);
    returninfo.mpam_sp = pspace;
    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)
    SecurityState security = SecurityStateAtEL(el);
    PARTIDspaceType space = PARTIDspaceFromSS(security);
    boolean use_default = !(HaveMPAMExt() && MPAMisEnabled());
    PARTIDspaceType altspace = AltPARTIDspace(el, security, space);
    space = altspace;
    if HaveEMPAMExt() && security == SS_Secure then
        if MPAM3_EL3.FORCE_NS == '1' then
            space = PIdSpace_NonSecure;
        if MPAM3_EL3.SDEFLT == '1' then
            use_default = TRUE;
    if !use_default then
        return genMPAM(el, InD, space);
    else
        return DefaultMPAMinfo(space);
```

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(bits(2) el, boolean InD)
    PARTIDtype partidel = getMPAM_PARTID(el, InD);
    PARTIDtype partid_max = MPAMIDR_EL1.PARTID_MAX;
    if UInt(partidel) > UInt(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(bits(2) 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

```
// 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 vpmrmax = UInt(MPAMIDR_EL1.VPMR_MAX);

    // vpartid_max is largest vpartid supported
    integer vpartid_max = (vpmrmax << 2) + 3;

    // One of many ways to reduce vpartid to value less than vpartid_max.
    if UInt(vpartid) > 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/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(bits(2) MPAMn, boolean InD)
    PMGtype pmg;
    boolean el2avail = EL2Enabled();

    if InD then
        case MPAMn of
            when '11' pmg = MPAM3_EL3.PMG_I;
            when '10' pmg = if el2avail then MPAM2_EL2.PMG_I else Zeros();
            when '01' pmg = MPAM1_EL1.PMG_I;
            when '00' pmg = MPAM0_EL1.PMG_I;
            otherwise pmg = PMGtype UNKNOWN;
    else
        case MPAMn of
            when '11' pmg = MPAM3_EL3.PMG_D;
            when '10' pmg = if el2avail then MPAM2_EL2.PMG_D else Zeros();
            when '01' pmg = MPAM1_EL1.PMG_D;
            when '00' 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 MOD 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 on page A-391 shows how partitions can have private and shared Portion Bit Map (PBM) allocations.

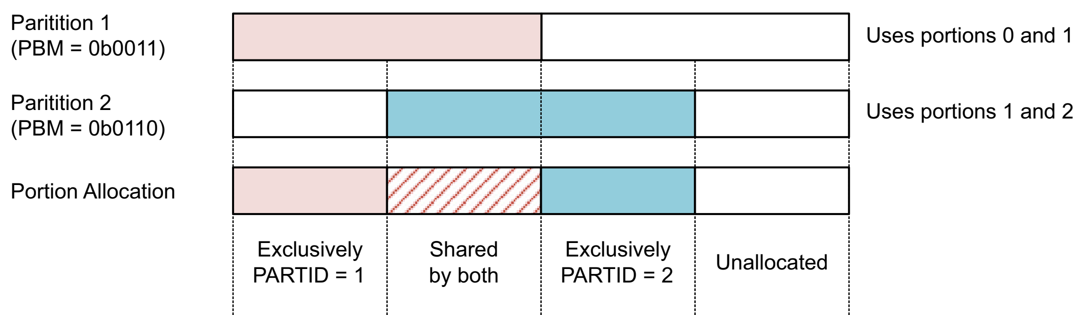


Figure A-1 Generic portion shared and exclusive allocations.

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 on page A-391. Each bit grants the PARTID using this control setting to allocate the portion corresponding to that bit.

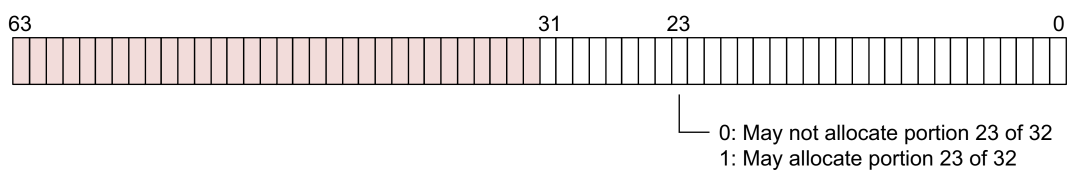


Figure A-2 Generic portion bit map.

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 9-189 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.
- 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.

B.4 Discovery of resource to RIS mapping

Software needs to know which RIS value to use to control a resource instance of the MSC.

This mapping is not available from MSC IDRs. It might be given as a firmware data table or other means beyond the hardware ID registers.

B.5 Discovery of wired interrupts

There are two interrupt sources in an MPAM MSC and they are replicated in the Secure and Non-secure MPAM behaviors. It is not possible to discover the connection of the four interrupts to GIC inputs from the MSC MPAM ID registers. This information must come from the firmware information.

Firmware must provide information on the connection and grouping of MPAM wired interrupts.

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 Armv8, for Armv8-A architecture profile*.

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.

ALTSP
Alternative PARTID space.

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.

Downstream Information propagating in the direction from Requesters towards terminating Completer components.

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 Requester. 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 resource 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 Requester to Completer, 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 Requesters, the security state can be determined in other ways.

MPAM_SP In MPAM for RME the MPAM PARTID space indication.

MSC Memory-system Component. See Memory-system component.

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.

RIS

Resource instance selection.

RME

Realm Management Extension. RME specifies how PE execution context is mapped to Security states.

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.

Upstream Information propagating in the direction from terminating Completer components towards Requesters.

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.